Declarative Languages for Machine Learning and Data Mining

Luc De Raedt

ICLP, 2015

with slides from

Angelika Kimmig, Davide Nitti, Siegfried Nijssen, Tias Guns, Paolo Frasconi, Francesco Orsini

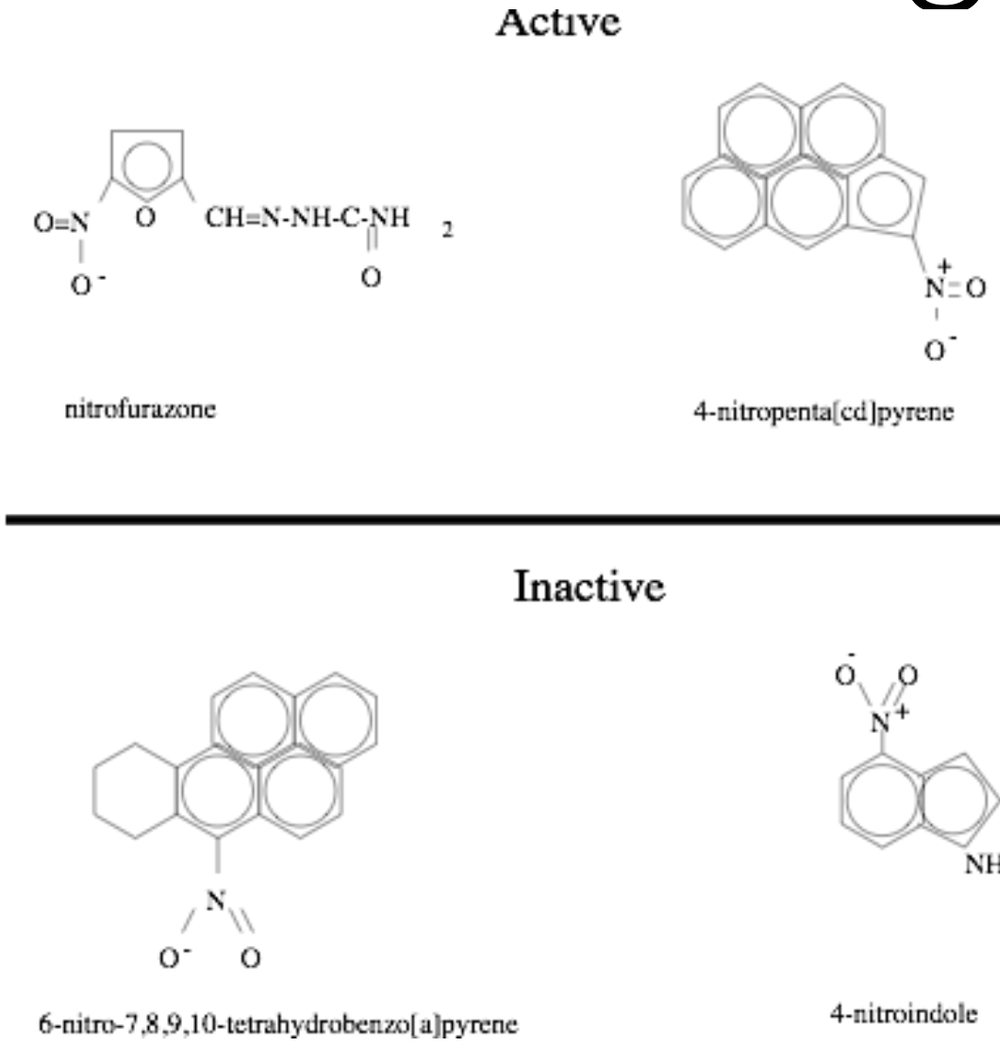
See also [AAAI 15 Senior Member Track Paper]

Inductive Logic Programming

- In 1995 :
  - focus on symbolic data and methods
  - focus on using and producing knowledge
  - learning programs typically written in Prolog

Inductive Logic Programming

[Srinivasan et al. AIJ 96]



Data = Set of Small Graphs

General Purpose
Logic Learning System

Uses and Produces
Knowledge

Learning from entailment

```
molecule(225).
logmutag(225,0.64).
lumo(225,-1.785).
logp(225,1.01).
nitro(225,[f1_4,f1_8,f1_10,f1_9]).
atom(225,f1_1,c,21,0.187).
atom(225,f1_2,c,21,-0.143).
atom(225,f1_3,c,21,-0.143).
atom(225,f1_4,c,21,-0.013).
atom(225,f1_5,o,52,-0.043).
...
ring_size_5(225,[f1_5,f1_1,f1_2,f1_3,f1_4]).
hetero_aromatic_5_ring(225,[f1_5,f1_1,f1_2,f1_3,f1_4]).
```

background

```
bond(225,f1_1,f1_2,7).
bond(225,f1_2,f1_3,7).
bond(225,f1_3,f1_4,7).
bond(225,f1_4,f1_5,7).
bond(225,f1_5,f1_1,7).
bond(225,f1_8,f1_9,2).
bond(225,f1_8,f1_10,2).
bond(225,f1_1,f1_11,1).
bond(225,f1_11,f1_12,2).
bond(225,f1_11,f1_13,1).
```

```
mutagenic(225), ...
```

examples

```
mutagenic(M) :- nitro(M,R1), logp(M,C), C > 1 .
```

rules

Machine learning and data mining

- Today
  - Big data — a lot more data available
  - Low-level and high-level features
  - focus on performance and scalability, less on understandability
  - kernels, probabilistic methods, constrained optimisation, statistics and linear algebra

Declarative methods

- There has been a paradigm shift in the field of AI from programming to solving (Hector Geffner at ECAI 2012)

- Use of solvers and declarative languages is common in AI
  - SAT, MAX-SAT, CSP, ASP, MDPs, ILP, MP, …

- Two sources of inspiration for this talk
  - Statistical learning (convex optimization) — Stephen Boyd
  - Constraint Programming in Practice — Helmut Simonis

Role of Declarative Methods in ML and DM ?

Role of Declarative Methods in ML and DM ?

Two aspects

Role of Declarative Methods in ML and DM ?

This talk:
a personal perspective, a possible answer

This talk

- Using and developing declarative languages for ML/DM — a survey
  - inductive query languages (database perspective)
  - modelling languages for ML/DM (constraint / answer set programming)
  - programming languages for ML/DM (programming language perspective)
    - probabilistic
    - kernel-based

Inductive Databases

Data Mining

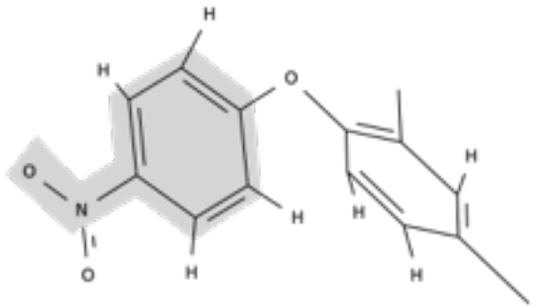
Given

- a database containing instances or transactions D
    - the set of instances
    - a hypothesis space or pattern language L
    - a selection predicate, query or set of constraints Q

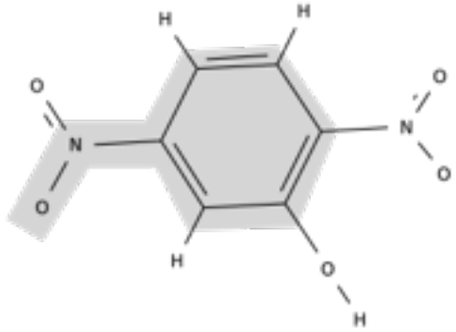
Find $Th(Q,L,D) = \{ h \in L \mid Q(h,D) = true \}$

[Mannila and Toivonen, 96]

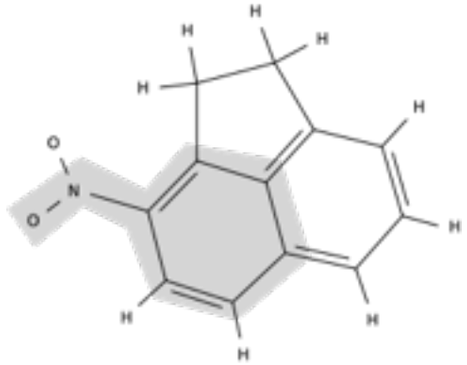
Pattern Mining



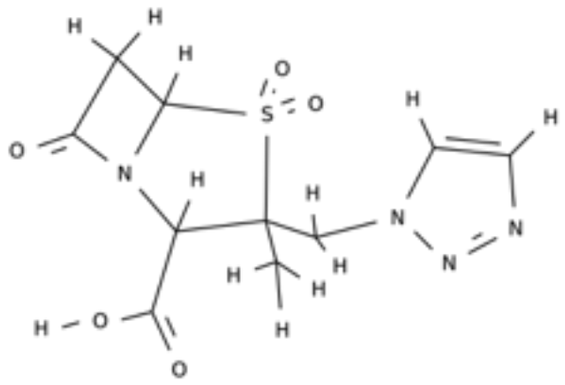
Mutagenic



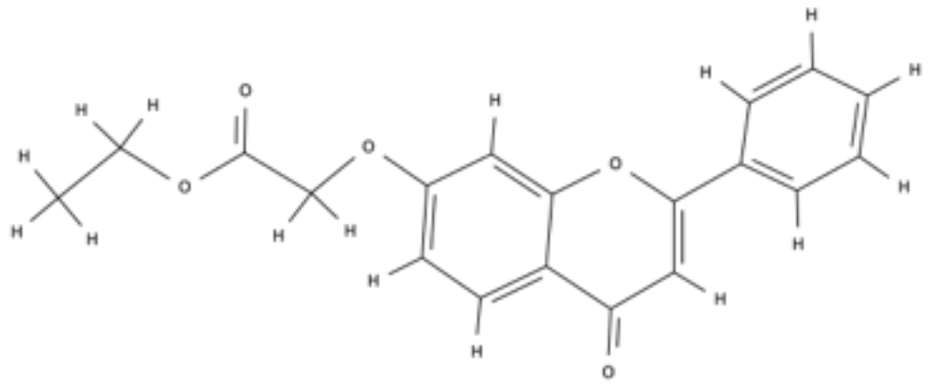
Mutagenic



Mutagenic



Clean



Clean

Find patterns that are frequent in mutagenic, infrequent in clean data.

Imielinski and Mannila (1995)

database perspective

The concept of data mining as a querying process.

- Make first class citizens of patterns.
- Query not only the data but also the patterns.
- Tightly integrate databases and data mining.
- Search for the equivalent of Codd's relational algebra for data mining

"From the user perspective, there is no such thing as a real discovery, just a matter of the expressive power of the query language."

An inductive database example
Virtual Mining Views (Blockeel et al. 12)

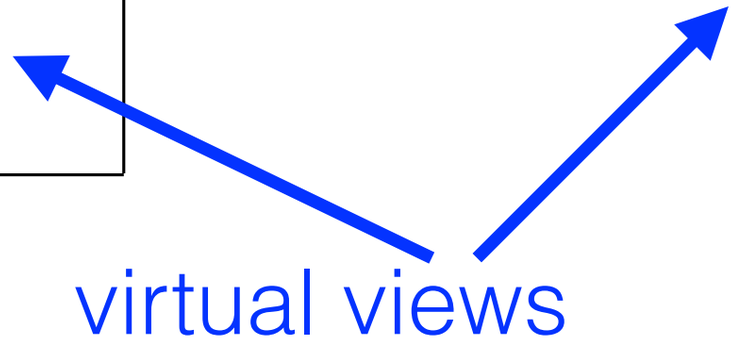
Beer	Brand	Color	Alcohol%
1	Westmalle Tripel	Blonde	9.5
2	Orval	Dark	6.2
3	Straffe Hendrik	Gold	9
4	Straffe Hendrik	Dark	11
....	...	...	...

An inductive database example
Virtual Mining Views (Blockeel et al. 12)

Beer	Brand	Color	Alcohol%
1	Westmalle Tripel	Blonde	9.5
2	Orval	Dark	6.2
3	Straffe Hendrik	Gold	9
4	Straffe Hendrik	Dark	11
....	...	...	...

Cid	Brand	Color	Alcohol%
c1	Westmalle Tripel	?	?
c2	Westmalle Tripel	Blonde	?
c3	?	Blonde	?
c4	?	Blonde	9.5
c5	Straffe Hendrik	?	?
....	...	...	...

cid	frequency	size
c5	2	1
...	...	...



An inductive database example
Virtual Mining Views (Blockeel et al. 12)

```
SELECT C.*, S.supp, S.sz,
S.supp * S.sz AS area
FROM BeerConcepts C, BeerSets S
WHERE (C.cid = S.cid AND (S.freq * S.sz > 60))
OR (S.freq > 10)
```

Cid	Brand	Color	Alcohol%
c1	Westmalle Tripel	?	?
c2	Westmalle Tripel	Blonde	?
c3	?	Blonde	?
c4	?	Blonde	9.5
c5	Straffe Hendrik	?	?
....	...	...	...

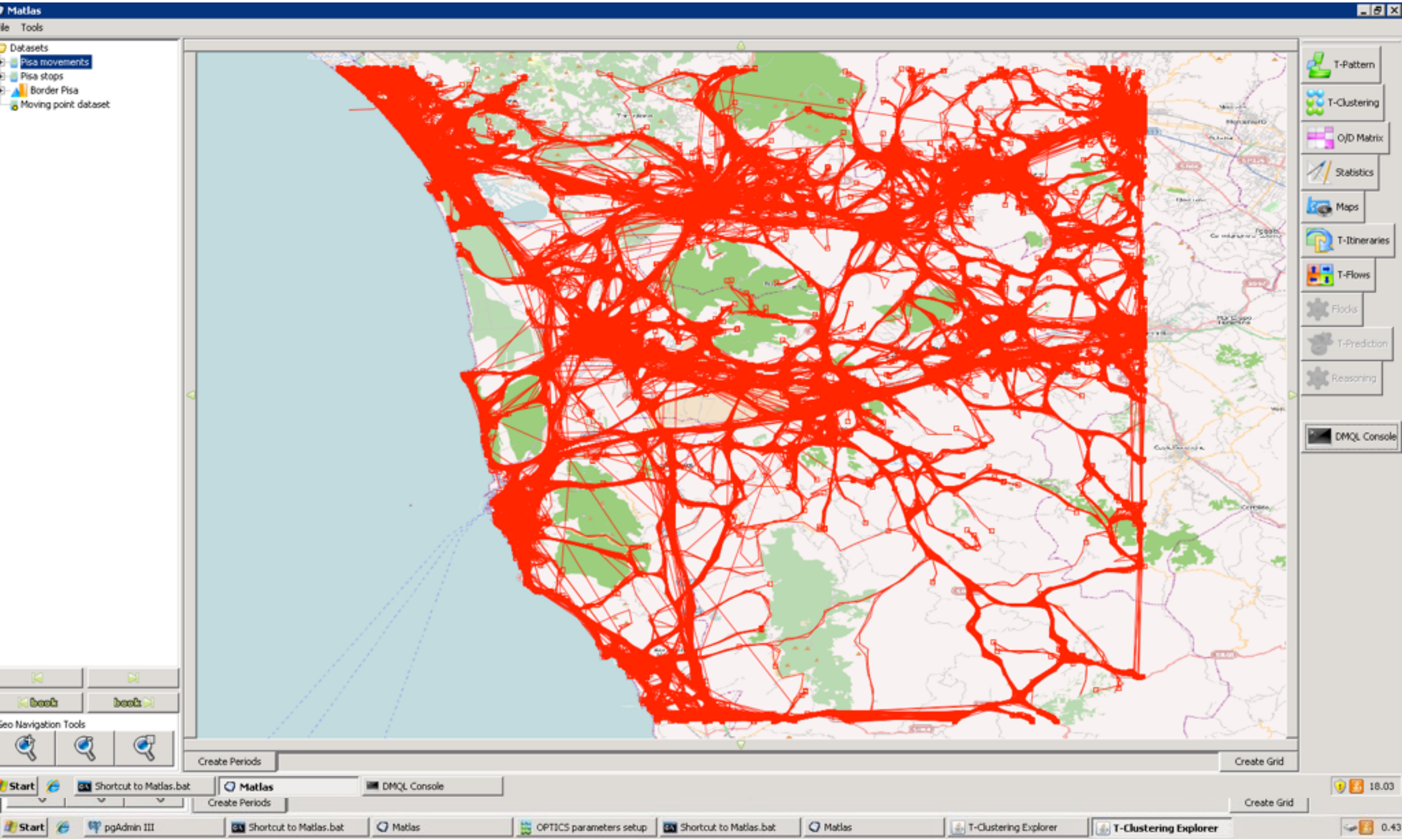
BeerConcepts

cid	frequency	size
c5	2	1
...	...	...

BeerSets

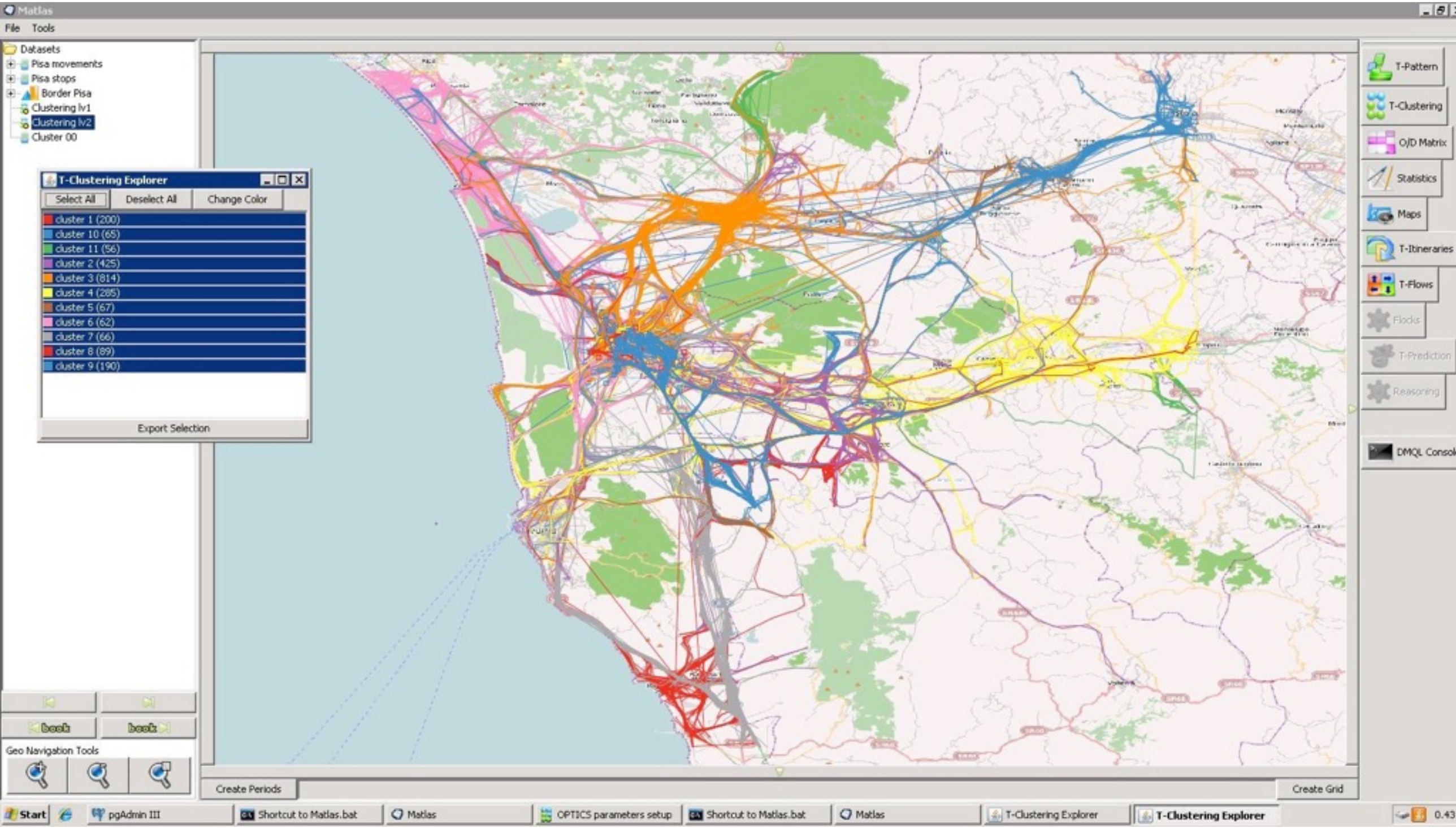
Slides courtesy
Dino Pedreschi

Understanding city access patterns



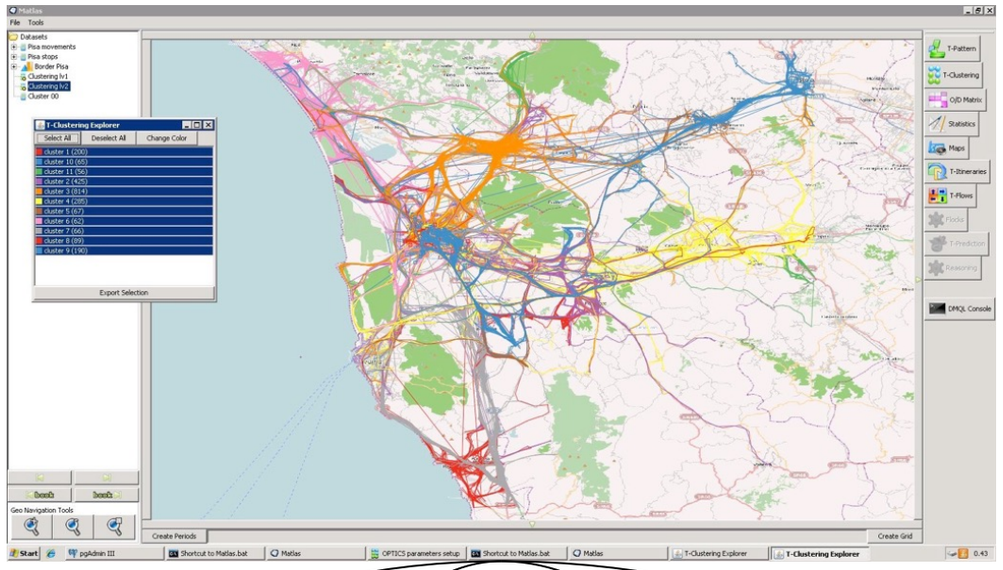
Slides courtesy
Dino Pedreschi

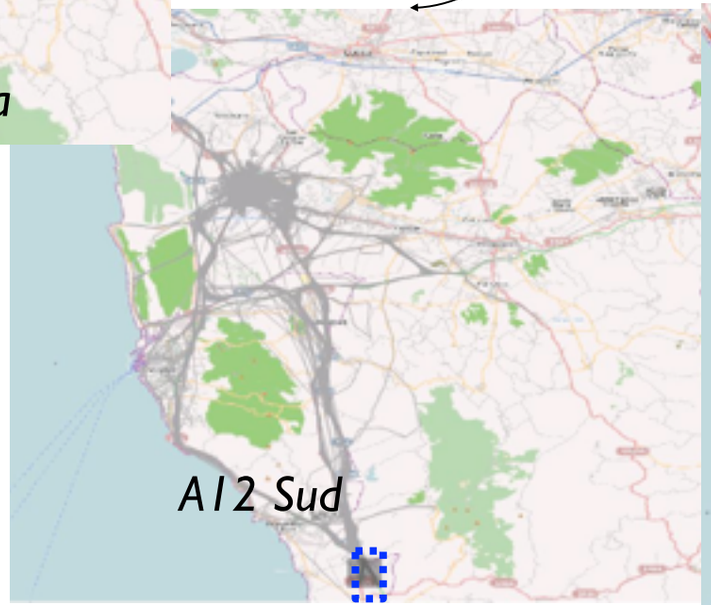
Understanding city access patterns

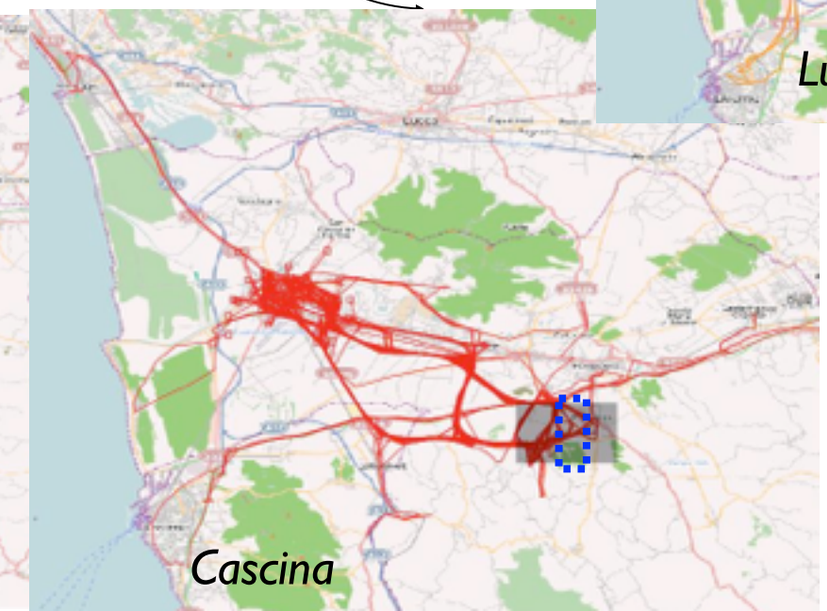


Slides courtesy
Dino Pedreschi

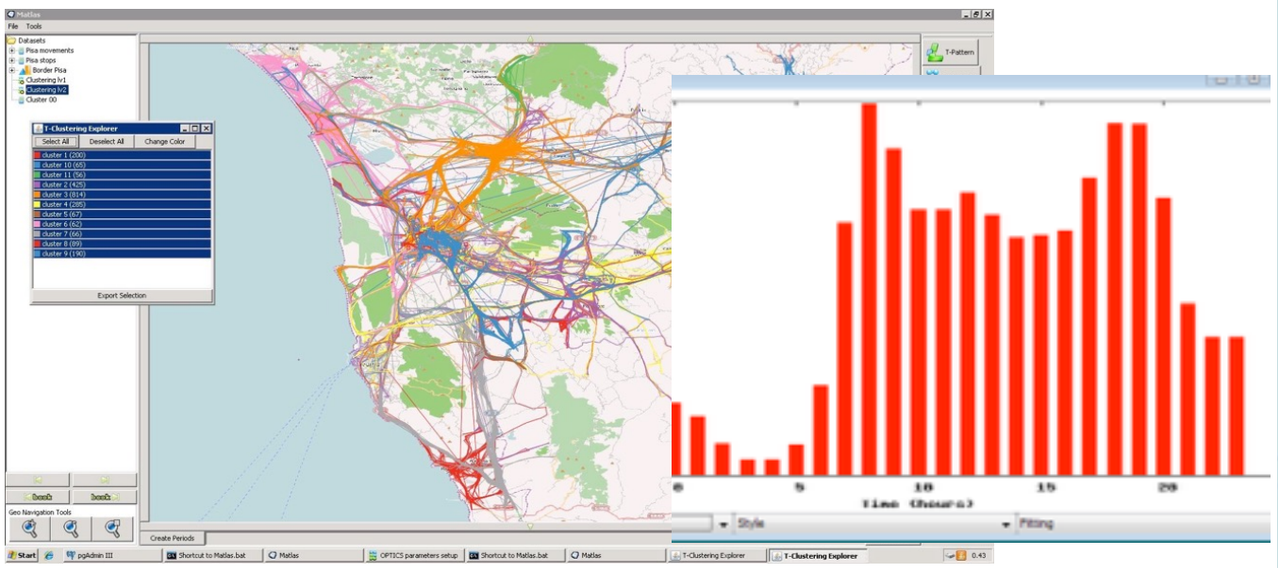
Understanding city access patterns

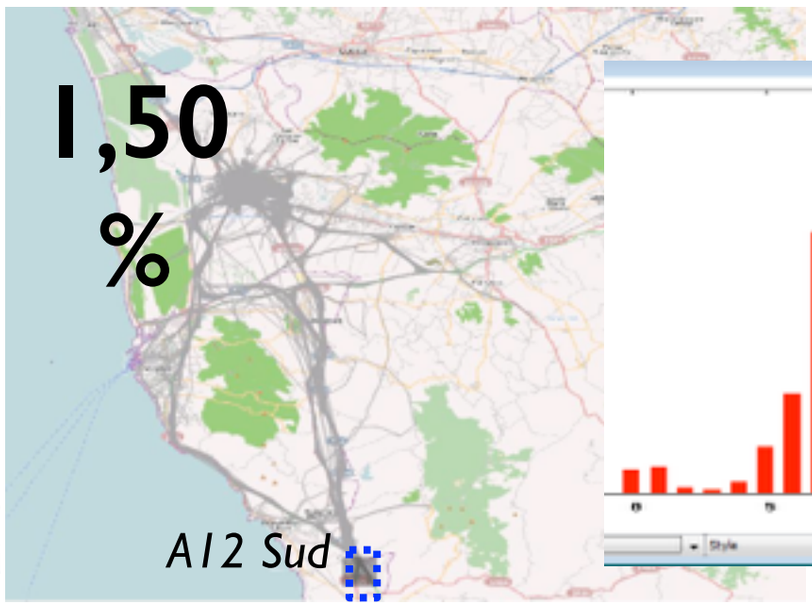


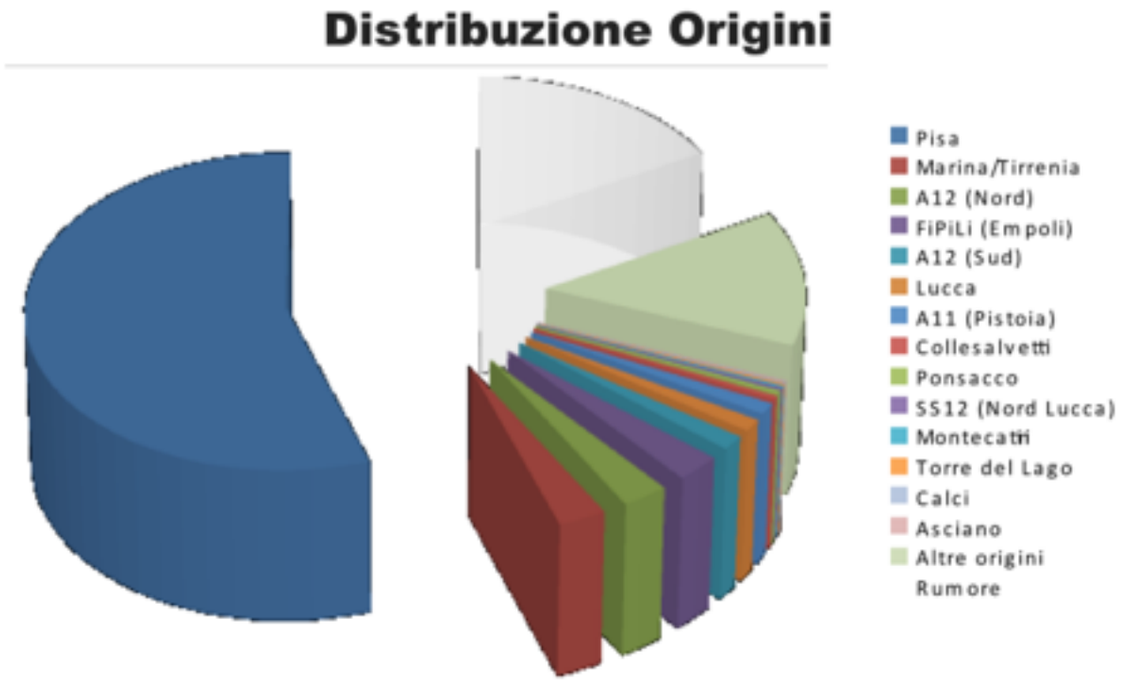


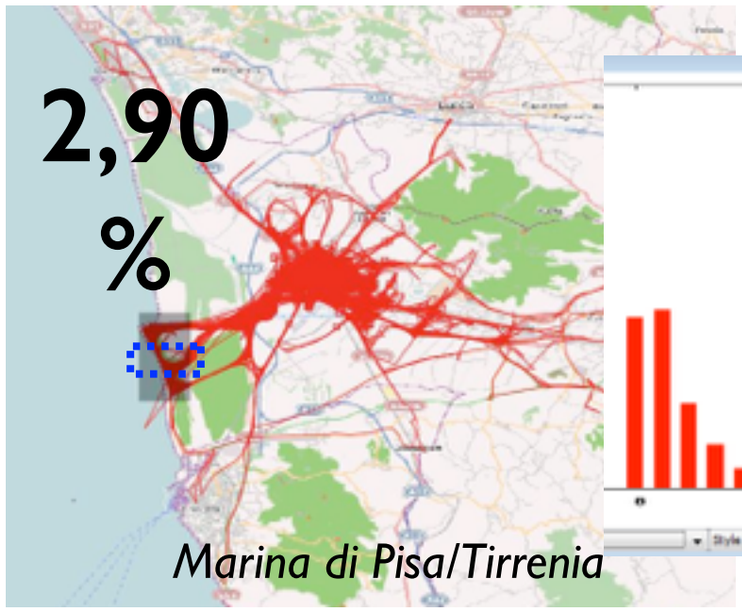


Slides courtesy
Dino Pedreschi



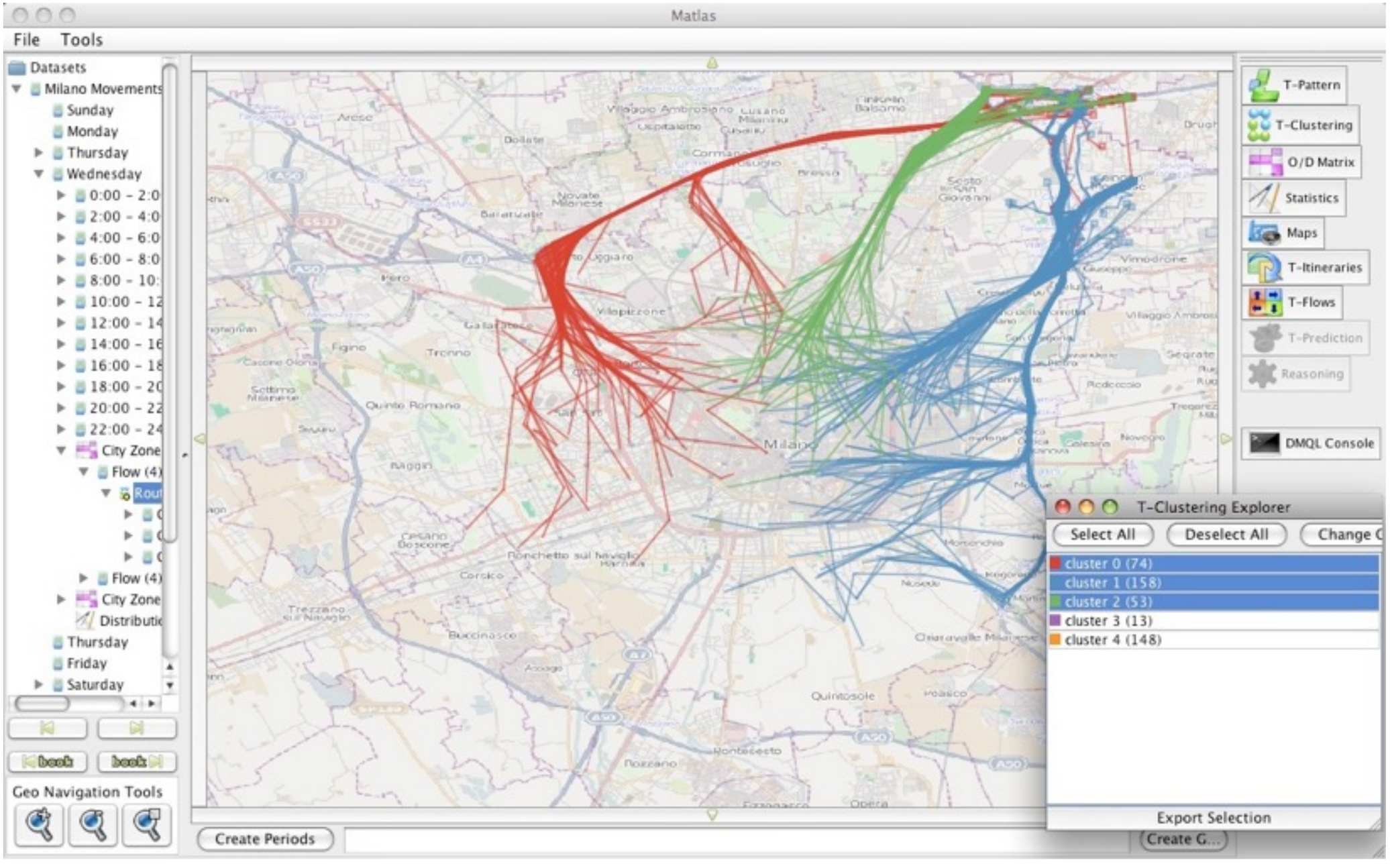






Slides courtesy
Dino Pedreschi

M-Atlas kdd.isti.cnr.it

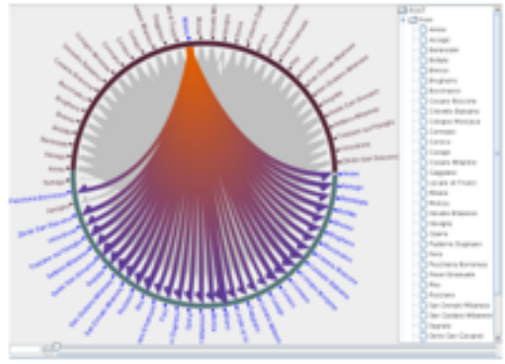


F. Giannotti, M. Nanni, D. Pedreschi, F. Pinelli, C. Renso, S. Rinzivillo, R, Trasarti.
Unveiling the complexity of human mobility by querying and mining massive trajectory data.
VLDB J., 2011

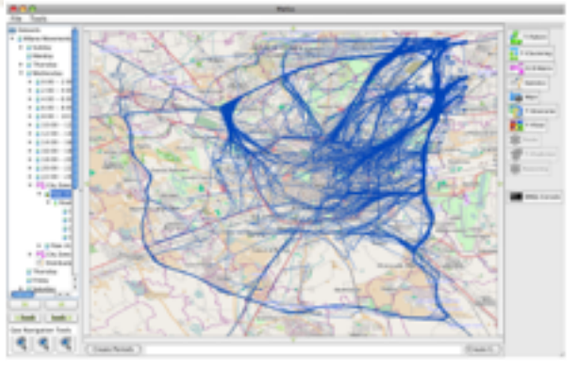
DMQL EXPRESSIVENESS:

How do people leave the city toward suburban areas?

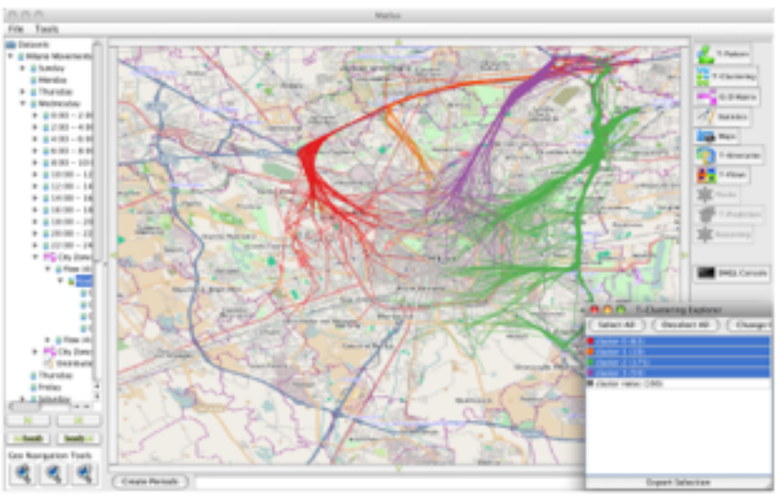
```
CREATE MODEL MilanODMatrix AS MINE ODMATRIX
FROM (SELECT t.id, t.trajectory FROM TrajectoryTable t),
(SELECT orig.id, orig.area FROM MunicipalityTable orig),
(SELECT dest.id, dest.area FROM MunicipalityTable dest)
```



```
CREATE RELATION CenterToNESuburbTrajectories USING ENTAIL
FROM (SELECT t.id, t.trajectory FROM TrajectoryTable t, MilanODMatrix m
WHERE m.origin = Milan AND
m.destination IN (Monza, ..., Brugherio))
```



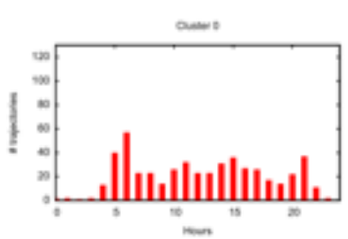
```
CREATE MODEL ClusteringTable AS MINE T-CLUSTERING
FROM (Select t.id, t.trajectory from CenterToNESuburbTrajectories t)
SET T-CLUSTERING.FUNCTION = ROUTE_SIMILARITY AND
T-CLUSTERING.EPS = 400 AND
T-CLUSTERING.MIN_PTS = 5
```

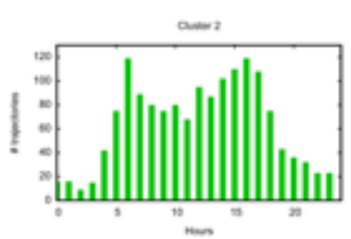


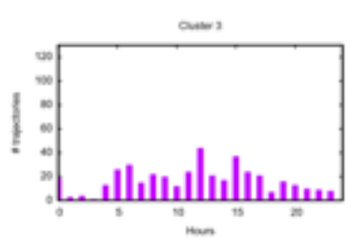
M-ATLAS -- *Gianotti et al.VLDB J. 1 1*

Mobility mining

Slides courtesy Pisa group







Inductive databases

- Many inductive query languages have been developed (supporting decision trees, complex pattern mining, clustering, geographical information systems …), e.g. MineRule (Meo), MSQL (Iemielinski), DMQL (Han), IQL (Nijssen), LDL ...

- How does it work :
    - integration in database system + query optimization + specific type of pattern;
    - usually: a call to an external “procedural” solver
- Challenges :
    - different implementations needed for different pattern types, no “universal” algebra for data mining known; the quest remains open
    - loose integration among different pattern types, sometimes more a toolbox
    - limited expressiveness (e.g. user-defined constraints absent)

Constraint-Based Pattern Mining

joint work with Tias Guns, Siegfried Nijssen et al.

Pattern Mining

A. frequent pattern

- which patterns are frequent ?

$$Th(\mathcal{L}, Q, \mathcal{D}) = \{p \in \mathcal{L} | Q(p, \mathcal{D}) = true\}$$

B. Correlated pattern mining = subgroup discovery

- which patterns are significant w.r.t. classes ? all patterns ? k-best patterns ?

$$Th(\mathcal{L}, Q, \mathcal{D}) = \arg_{p \in \mathcal{L}} \max_k \phi(p, \mathcal{D})$$

C. pattern set mining

- which pattern *set* is the *best* concept-description for the actives ? for the inactives ?

$$Th(\mathcal{L}, \mathcal{Q}, \mathcal{D}) = \{P \subseteq \mathcal{L} | \mathcal{Q}(P, \mathcal{D}) = true\}$$

Constraint-Based Mining

- Numerous constraints have been used
- Numerous systems have been developed
- And yet,
  - new constraints often require new implementations
  - very hard to combine different constraints

Constraint and Answer Set Programming

- Exists since about 20 ? years
- A general and generic methodology for dealing with constraints across different domains
- Efficient, extendable general-purpose systems exist, and key principles have been identified
- Surprisingly CP & ASP had not been used for data mining ?
- CP and ASP systems often more elegant, more flexible and more efficient than special purpose systems
- Also true for Data Mining ?

Itemset mining

Data

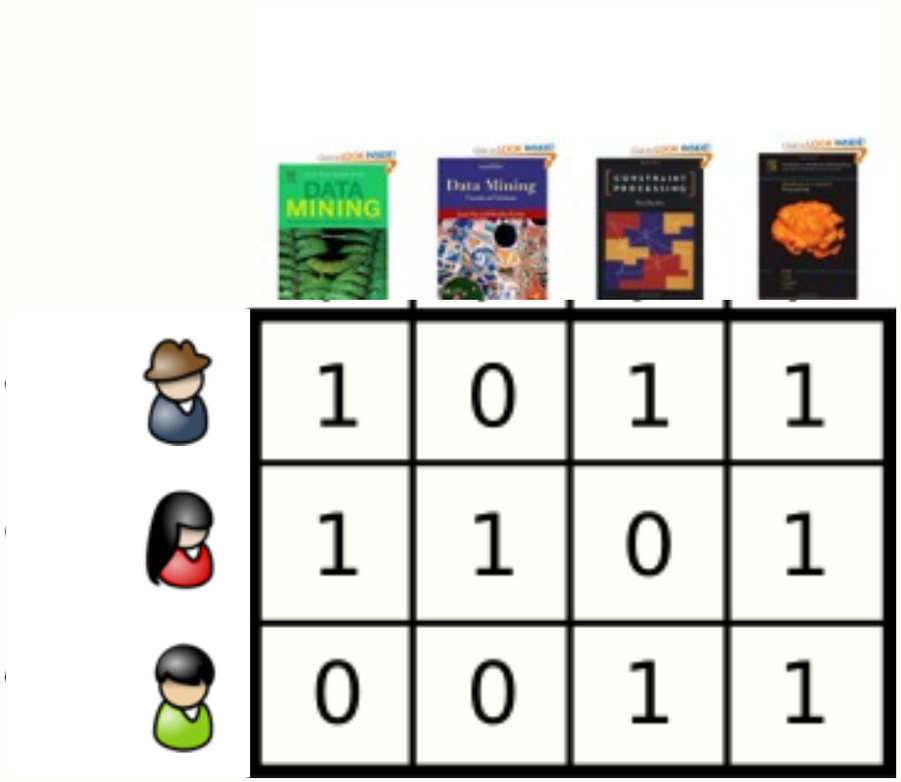
Frequent patterns

4

2

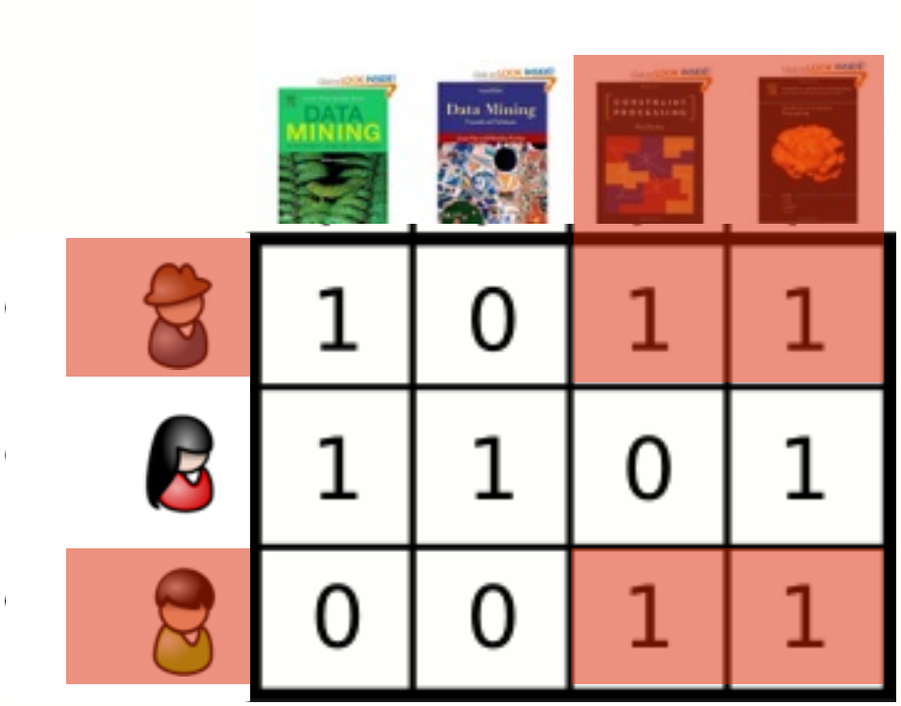
3

Frequent Itemset Mining



Frequent Itemset Mining

Find: all sets of items appearing frequently



cover(,) = { , }

frequency( ) = | { , } | = 2

Frequent Itemset Mining

Given

- $\mathcal{I} = \{1, \cdots, NrI\}$
  set of items

- $\mathcal{T} = \{1, \cdots, NrT\}$
  set of transactions

- $\mathcal{D} = \{(t, I) | t \in \mathcal{T}, I \subseteq \mathcal{I}\}$
  dataset

- $Items \subseteq \mathcal{I}$ and $Trans \subseteq \mathcal{T}$

Find $Items$ such that
$\mid covers(Items, \mathcal{D}) \mid > freq$
where $covers(Items, \mathcal{D}) =$
$\{t \in \mathcal{T} \mid (t, I) \in \mathcal{D}$ and $Items \subseteq I\}$

Frequent Itemset Mining

Given

- $\mathcal{I} = \{1, \cdots, NrI\}$
  set of items
- $\mathcal{T} = \{1, \cdots, NrT\}$
  set of transactions
- $\mathcal{D} = \{(t, I) | t \in \mathcal{T}, I \subseteq \mathcal{I}\}$
  dataset
- $Items \subseteq \mathcal{I}$ and $Trans \subseteq \mathcal{T}$

Find $Items$ such that
$| covers(Items, \mathcal{D}) | > freq$
where $covers(Items, \mathcal{D}) =$
$\{t \in \mathcal{T} \mid (t, I) \in \mathcal{D} \text{ and } Items \subseteq I\}$

```
int: Freq;
int: NrI;
int: NrT;

array[1..NrT] of set of 1..NrI: D;

var set of 1..NrI: Items;
var set of 1..NrT: Trans;

constraint card(Trans) > Freq;
constraint Trans = covers(Items, D);

solve satisfy;

function var set of int: cover(Items, D) =
let {
        var set of int: Trans,
        constraint forall (t in ub(Trans))
        (t in Trans ↔ Items subset D[t])
} in Trans;
```

MiningZinc [Guns et al IJCAI 13, ICDM 13]

Declarative Modeling

- Language goals:
  - high-level notation (similar to paper definitions)
  - solver-independent
  - user-defined abstractions

- mathematical-like language (Zinc)

- many solvers

- can define custom predicates ***and functions***

=> MiningZinc

[T. Guns, A. Dries, S. Nijssen, G. Tack, L. De Raedt, IJCAI 2013]

Frequent Itemset Mining

in IDP (Denecker et al.)

```
vocabulary FrequentItemsetMiningVoc {
  type Transaction
  type Item
  Freq: int
  Includes(Transaction,Item)
  FrequentItemset(Item)
}
theory FrequentItemsetMiningTh: FrequentItemsetMiningVoc {
  #{t: !i: FrequentItemset(i) => Includes(t,i) } >= Freq.
}
structure Input : FrequentItemsetMiningVoc {
  Freq = 7 // threshold for frequent itemsets
  Transaction = { t1; ... ; tn } // n transactions
  Item = {i1 ; ... ;  im }       // m items
  Includes = {t1,i2; t1,i7; ...} // items of transactions
}
```

FrequentItemset represents a set of items

FreqItemset must fulfill
#{t: FreqItemset ⊆ t} >= Freq.

in ASP — See Järvisalo, LPNMR 11

Closed Itemset Mining

```
int: Freq;
int: NrI;
int: NrT;

array[1..NrT] of set of 1..NrI: D;

var set of 1..NrI: Items;
var set of 1..NrT: Trans;
constraint card(Trans) > Freq;
constraint Trans = covers(Items, D);
constraint Items = cover_inv(Trans, D);
solve satisfy;
```

```
function var set of int: cover_inv(Trans,D)=
 let {
        var set of int: Items,
        constraint forall (i in ub(Items))
      (i in Items ↔ Trans subset D'[i] )
} in Items;
```

```
function var set of int: cover(Items, D) =
let {
        var set of int: Trans,
        constraint forall (t in ub(Trans))
      (t in Trans ↔ Items subset D[t] )
} in Trans;
```

Generality

	LCM [15]	MAFIA [6]	ExAMiner [4]	DualMiner [5]	DDPrune [12]	CP4IM
Constraints on data						
Minimum frequency	X	X	X	X		X
Maximum frequency				X		X
Emerging patterns						X
Condensed Representations						
Maximal	X	X		X		X
Closed	X	X				X
$\delta-$Closed						X
Constraints on syntax						
Max/Min total cost			X	X		X
Minimum average cost			X			X
Max/Min size	X	X	X	X		X
Constraints on labelled data						
Minimum correlation					X	X
Maximum correlation						X

[L. De Raedt, T. Guns, S. Nijssen, AAAI 2010]

Manual Encoding in Zinc

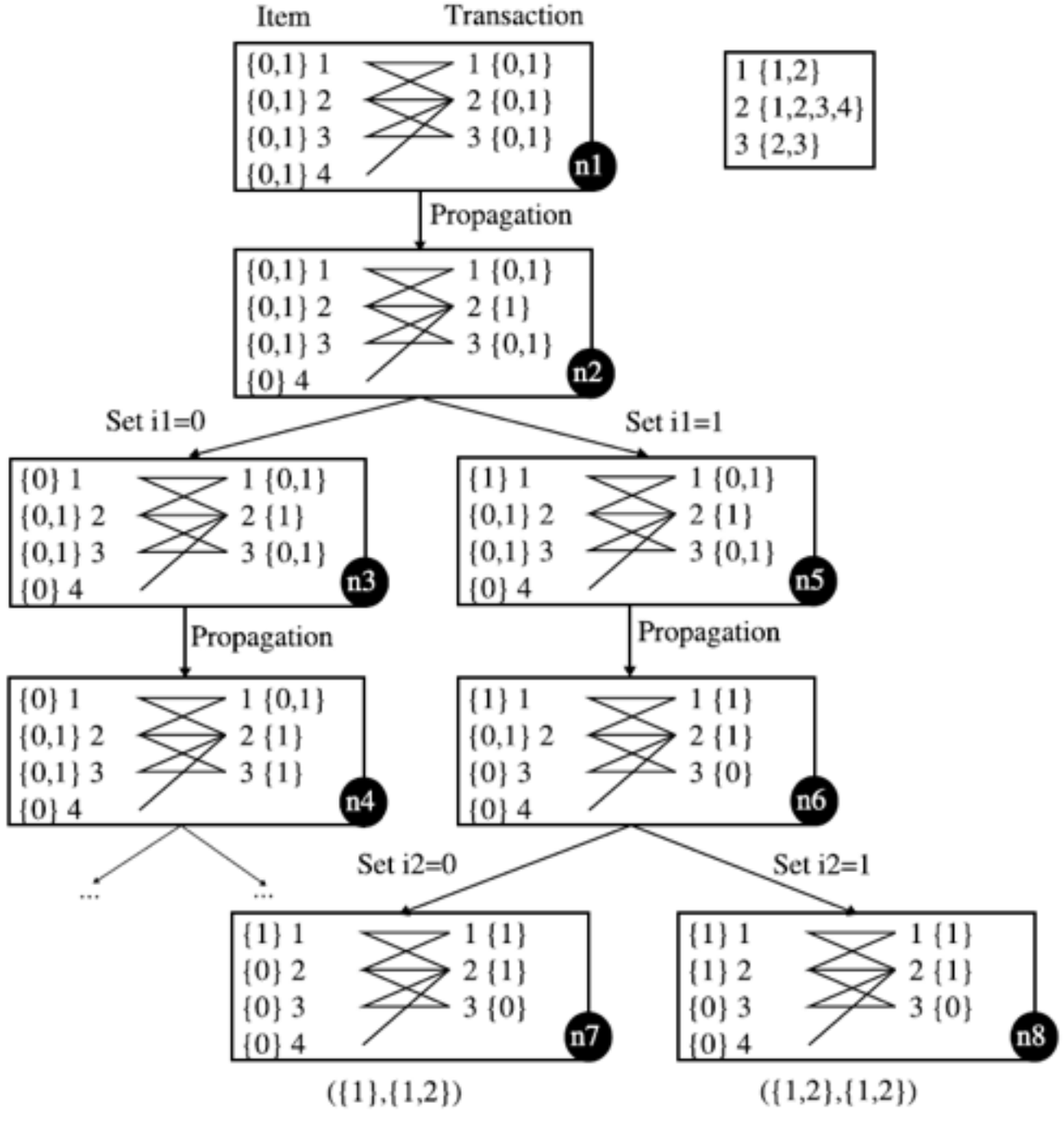
```
int: Freq;
int: NrI; int: NrT;
array [1..NrT] of set of int: D;

array [1..NrI] of var bool: Items;
array [1..NrT] of var bool: Trans;

constraint % encode D: every Trans complement has no supported Items
  forall(t in 1..NrT) (
     Trans[t] <-> sum(i in 1..NrI) ( Items[i]*(1 - (i in D[t])) ) <= 0
  );

constraint % frequency: every Item is supported by sufficently many Trans
  forall(i in 1..NrI) (
     Items[i] -> sum(t in 1..NrT) ( Trans[t]*(i in D[t]) ) >= Freq
  );

solve satisfy;
```

similar to original encodings, KDD 08,09

$$\forall t : T_t = 1 \Leftrightarrow \sum_i I_i(1 - D_{ti}) = 0$$

$$\sum T_t \geq minsup \quad \textbf{iff} \quad \forall i : I_i = 1 \Rightarrow \sum T_t D_{ti} \geq minsup$$

Resulting Search Strategy akin to Zaki's Eclat [KDD 97]



Discriminative Pattern Mining

```
int: NrI; int: NrT; int: Freq;
array[1..NrT] of set of int: D;
set of int: pos; set of int: neg;

var set of 1..NrI: Items;
var set of 1..NrT: Trans;

constraint Trans = cover(Items, D);

solve maximize
      card(Trans intersect pos) – card(Trans intersect neg) neg);
```

accuracy

Alternative opt. functions, for example:

```
solve maximize chi2(Trans, pos, neg);
```

with:

```
function float: chi2(Trans, pos, neg)
```

Correlation function

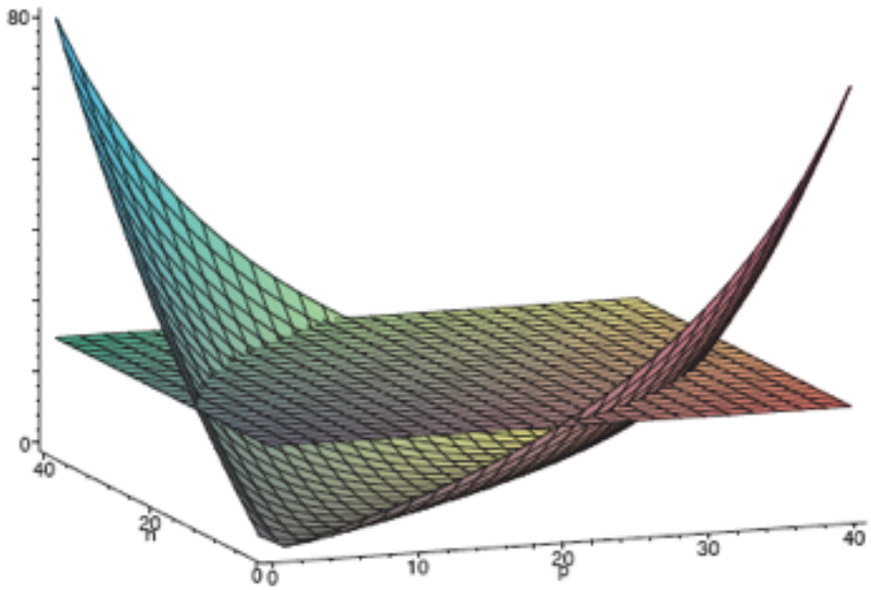
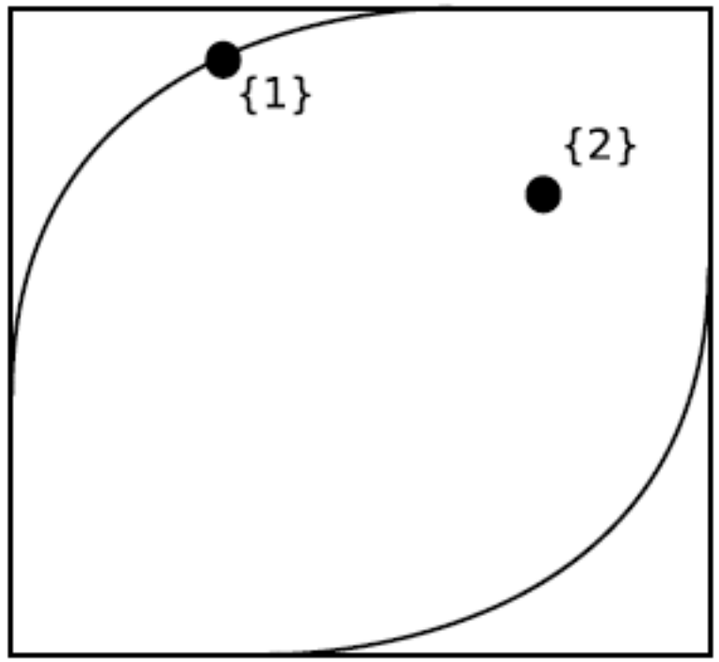


Figure 1: A plot of the χ^2 scoring function, and a threshold on χ^2.



Declarative Modeling

- Language goals:
  - high-level notation (similar to paper definitions)
  - solver-independent
  - user-defined abstractions

- mathematical-like language (Zinc)

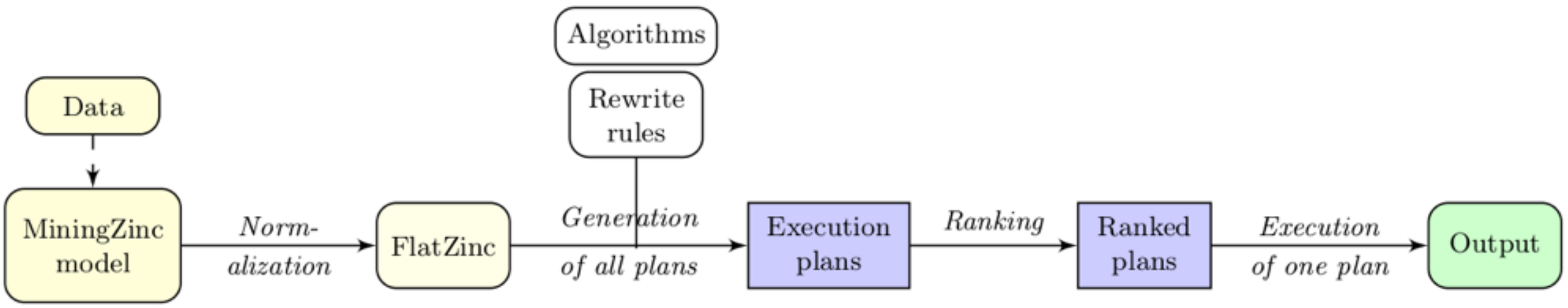
- many solvers

- can define custom predicates ***and functions***

=> MiningZinc

[T. Guns, A. Dries, S. Nijssen, G. Tack, L. De Raedt, IJCAI 2013]

Toolchain



1) Normalize to FlatZinc *(do not flatten lib_itemsetmining.mzn yet)*

2) Apply rewrite rules to:

    1) add redundant constraints

    2) detect (partial) applicability of specialised algorithms

    3) tailor to constraint solvers

3) Collect all feasible rewrite combinations = execution plans

4) Heuristically rank + execute a plan

```
var set of 1..NrI: Items; array[int] of set of int: TDB;
constraint card(cover(Items, TDB)) >= 20;
constraint card(cover(Items, TDB)) =< 40;
solve satisfy;
```

Three categories of execution plans:

A) Specialised algorithms only

- Eclat-maxfreq(TDB, 20, 40)
- LCMv2(TDB, 20) + maxcover(Items, TDB, 40)

B) Hybrid decomposition

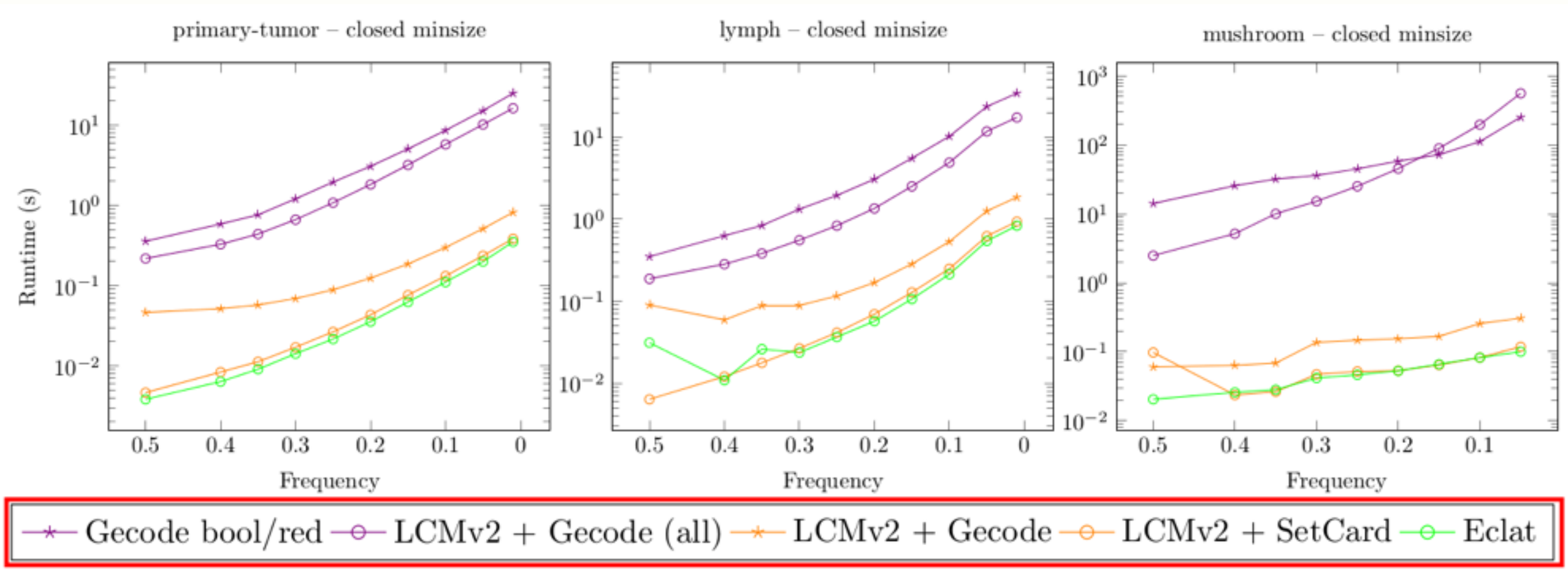
- LCMv2(TDB, 20) + gecode(card(cover(Items, TDB)) =< 40)
- LCMv2(TDB, 20) + frequency(Items, TDB, S) + gecode(S =< 40)

C) Generic solvers only

- gecode(...)
- gecode-bool(...)
- gecode-bool(... *+ redundant*)
- or-tools-bool(...)

Experiments, hybrid solving

frequent itemset mining, with minimum size and closure constraint



specialised solvers are much more efficient

cis-regulatory module detection

Genomic sequences

Seq_1	(1,10) (65,74)	(12,20) (80,88)	(43,51) (53,61) (90,98)	(72,78)
Seq_2	(33,42) (85,94)		(49,57) (56,64) (91,99)	(50,56) (52,58)
Seq_3	(1,10) (82,91)	(45,53) (58,66) (75,83) (90,88)	(24,32) (72,80) (89,97)	

regulators

with CP: add domain-specific constraints

[H. Sun, T. Guns, et al, Nucleic Acids Research 2012]

Declarative Pattern Mining

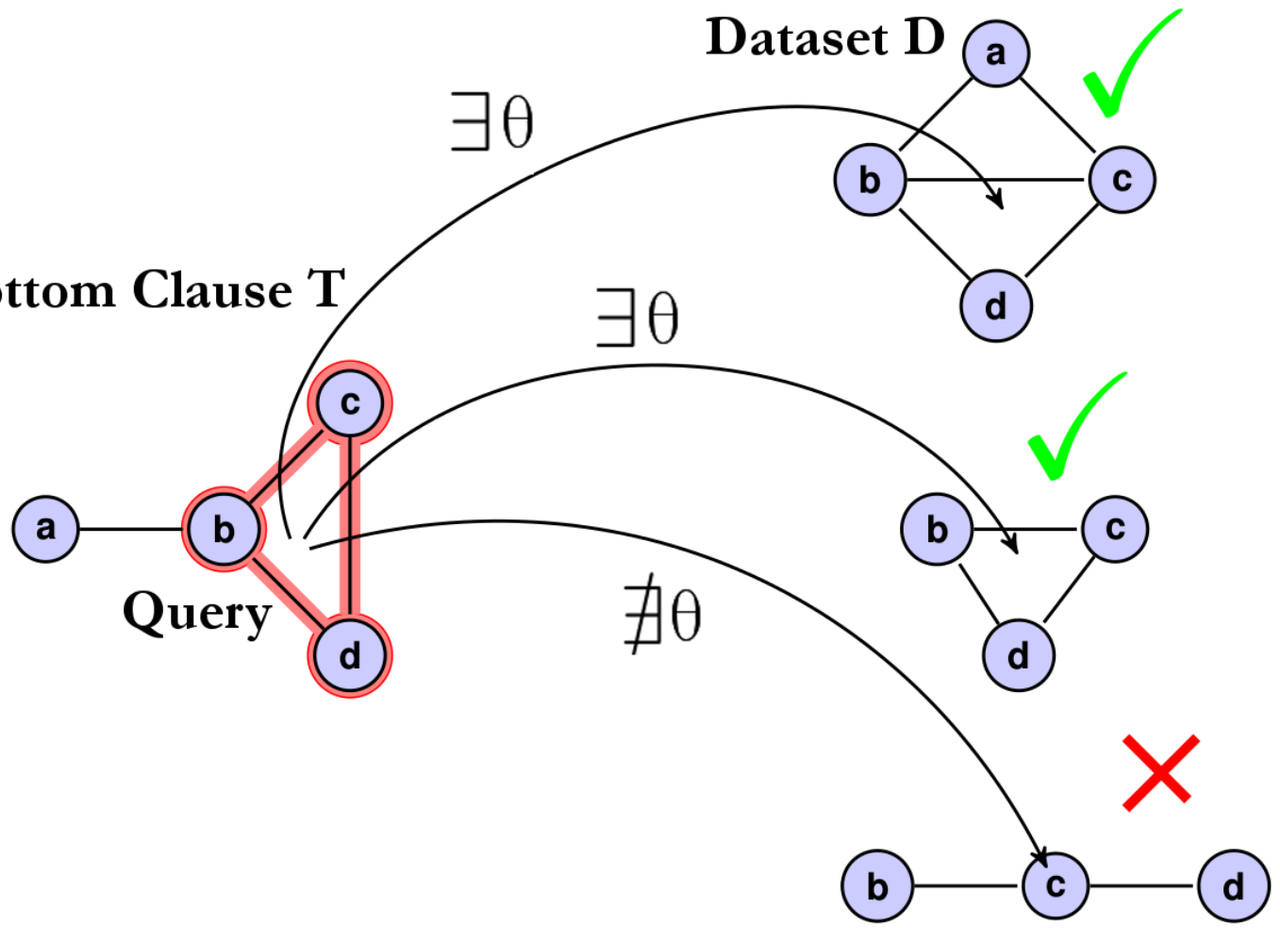
- Quite some work on data mining and CP
  - pattern mining : P. Boizumault, B. Cremilleux, L. De Raedt, T. Guns, S. Jabbour, M. Jarvisalo, S. Loudni, S. Nijssen, B. O'Sullivan, W. Ugarte, A. Kemmar, ….
  - clustering : B. Babaki, I. Davidson, T.B.H. Dao, O. du Merle, S. Gilpin, P. Hansen, S. Nijssen, C. Vrain, …
  - Several papers at CP 15
- Still many limitations and challenges
  - Expressive power of modeling tools
  - Efficient execution / compilation

Mining graphs

- Three problems
  - subgraph isomorphism — NP-complete subproblem
  - enumeration of subgraphs
    - requires lower bound
      - canonical form (do not generate the same graph more than once)
  - independent subproblems reusing previous results;
- Some emerging approaches :
  - sequences [Negrevergne, CPAIOR 15],[Kemmar, CP 15],
  - graphs : [Paramonov, ILP15]
  - require clever encodings — the “programming part” of ASP / CP

Mining graphs

in IDP, [Paramonov ILP 95]



Multiple Graph Homomorphism Check:

https://dtai.cs.kuleuven.be/CP4IM/download.php

Apps | Myroute | My Touring Mobilis | Street Photography | Save to Mendeley | Google Scholar | dupong

CP4IM

Constraint Programming for Itemset Mining

Home | Software | Datasets | Publications

Software

MiningZinc, this high-level declarate framework for constraint-based mining is available on a separate page.

Download FIM_CP

The frequent and constraint-based itemset mining system using Constraint Programming.

- Publication: *L. De Raedt, T. Guns, S. Nijssen. Constraint programming for itemset mining, KDD 2008.*
- Publication: *T. Guns, S. Nijssen, L. De Raedt. Itemset mining: A constraint programming perspective, Artificial Intelligence 175(12-13), 2011.*
- **Latest version:** fimcp-2.7.tar.gz
- License: MIT license
- Language: C++ (tested on Linux, known to work under Mac and Windows)
- Latest changes: Release with new, easier, build system: it now automatically downloads, configures and compiles the Gecode CP solver (version 3.7.1).

Installation instructions in the accompagnying README. See online usage instructions and examples.

inductiveconstraints.eu/miningzinc/#

Click to go back, hold to see history

uring Mobilis | Street Photography | Save to Mendeley | Google Scholar | dupong

MiningZinc | Basic usage | Modeling | Examples | Download

Basic usage

MiningZinc can be used through two interfaces: as a command line tool and as a Python package.

From the command line

MiningZinc can be run through its command line interface. There are four modes:

- `list` : analyze the model and list the possible execution plans
- `solve` : solve the model
- `interactive` : combination of previous two modes with a choice menu
- default: solve the model using the first available strategy

The default usage of MiningZinc is:

```
./miningzinc model.mzn data1.dzn data2.dzn -D "Param1=10;"
```

Programming Languages for Machine Learning

Programming Languages for Machine Learning

Probabilistic Programming

with Davide Nitti, Angelika Kimmig, Bogdan Moldovan

Tom Mitchell

Can we design programming languages containing machine learning primitives?

Can a new generation of computer programming languages directly support writing programs that learn?

Why not design a new computer programming language that supports writing programs in which some subroutines are hand-coded while others are specified as “to be learned.” Such a programming language could allow the programmer to declare the inputs and outputs of each “to be learned” subroutine, then select a learning algorithm from the primitives provided by the programming language.

Tom Mitchell

Can we design programming languages containing machine learning primitives?

Can a new generation of computer programming languages directly support writing programs that learn?

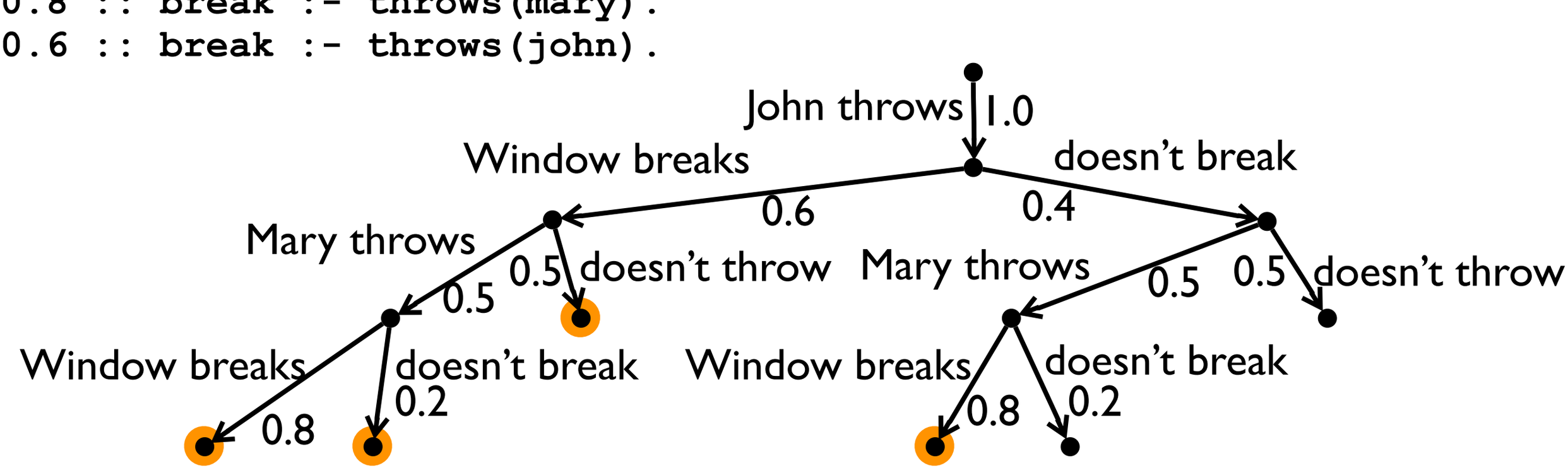
Why not design a new computer programming language that supports writing programs in which some subroutines are hand-coded while others are specified as “to be learned.” Such a programming language could allow the programmer to declare the inputs and outputs of each “to be learned” subroutine, then select a learning algorithm from the primitives provided by the programming language.

[Vennekens et al, ICLP 04]

Probabilistic Prologs
CP-Logic & ProbLog

```
throws(john).
0.5::throws(mary).

0.8 :: break :- throws(mary).
0.6 :: break :- throws(john).
```

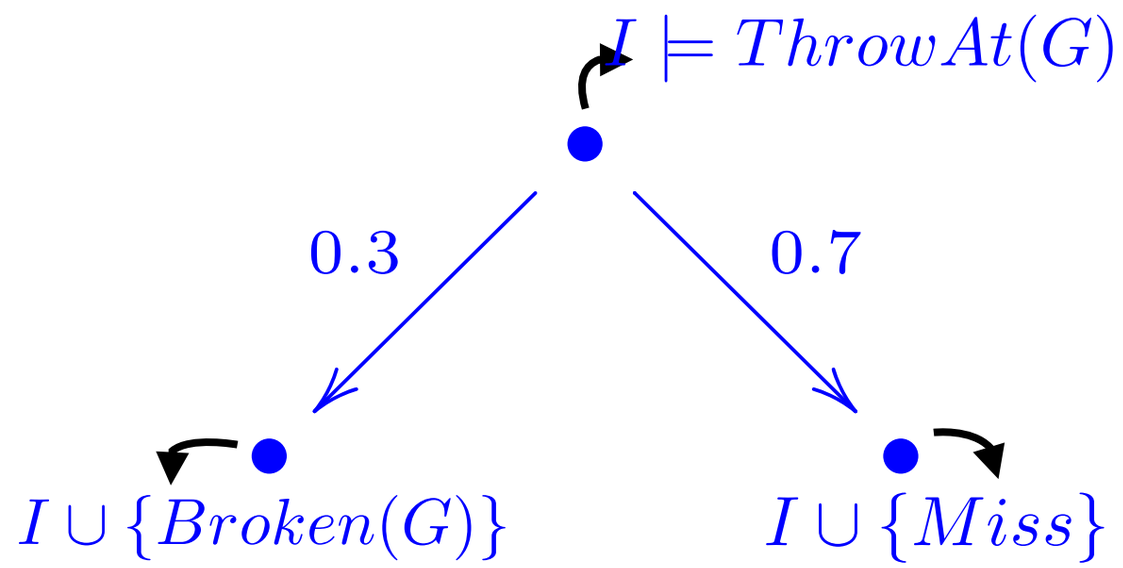
probabilistic causal laws



$$P(break)=0.6\times0.5\times0.8+0.6\times0.5\times0.2+0.6\times0.5+0.4\times0.5\times0.8$$

marginal probability of ?- break.

Semantics



(Broken(G) 0.3) ∨ (Miss 0.7)
← ThrowAt(G)

Probability tree is an execution model of theory iff:

- Each tree-transition matches causal law
- The tree cannot be extended

Each execution model defines the same probability distribution over final states

Slides CP-logic courtesy Joost Vennekens

Distributional Clauses (DC)

closely related to BLOG [Russell et al.]

Discrete- and continuous-valued random variables

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

closely related to BLOG [Russell et al.]

Discrete- and continuous-valued random variables

random variable with Gaussian distribution

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
```

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

closely related to BLOG [Russell et al.]

Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
    ≃length(OBot) ≥ ≃length(OTop),
    ≃width(OBot) ≥ ≃width(OTop).
```

comparing values of random variables

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

closely related to BLOG [Russell et al.]

Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
      ≃length(OBot) ≥ ≃length(OTop),
      ≃width(OBot) ≥ ≃width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                            0 : pitcher, 0.8676 : plate,
                            0.0284 : bowl, 0 : serving,
                            0.1016 : none])
                          :- obj(Obj), on(Obj,O2), type(O2,plate).
```

random variable with discrete distribution

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

closely related to BLOG [Russell et al.]

Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
      ≃length(OBot) ≥ ≃length(OTop),
      ≃width(OBot) ≥ ≃width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                            0 : pitcher, 0.8676 : plate,
                            0.0284 : bowl, 0 : serving,
                            0.1016 : none])
                   :- obj(Obj), on(Obj,O2), type(O2,plate).
```

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

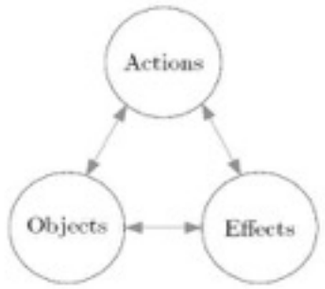
- Defines a generative process (as for CP-logic)
  - Tree can become infinitely wide, so exact inference infeasible and sampling needed; likelihood weighting or MCMC …
  - Well-defined under reasonable assumptions; see Gutmann et al., TPLP 11; Nitti et al., IROS 13, ICRA 14;
- Typical inference tasks :
  - marginal probability of a query P(query)
  - conditional probability P(query | evidence)
  - Bayesian or EM-based learning —- learning parameters and/or structure

Inference in PLP

- As in Prolog and logic programming
  - proof-based, using knowledge compilation
- As in Answer Set Programming
  - model based, using knowledge compilation
- As in Probabilistic Programming
  - sampling; uncommon in declarative methods

Learning relational affordances

Learn probabilistic model



Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

Learning relational affordances between two objects (learnt by experience)

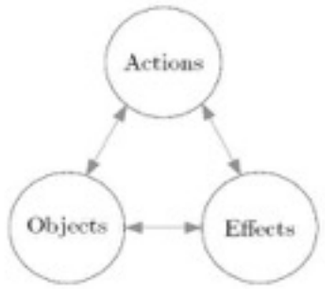
From two object interactions

Generalize to N

***Moldovan** et al. ICRA 12, 13, 14, PhD 15*

Learning relational affordances

Learn probabilistic model



Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

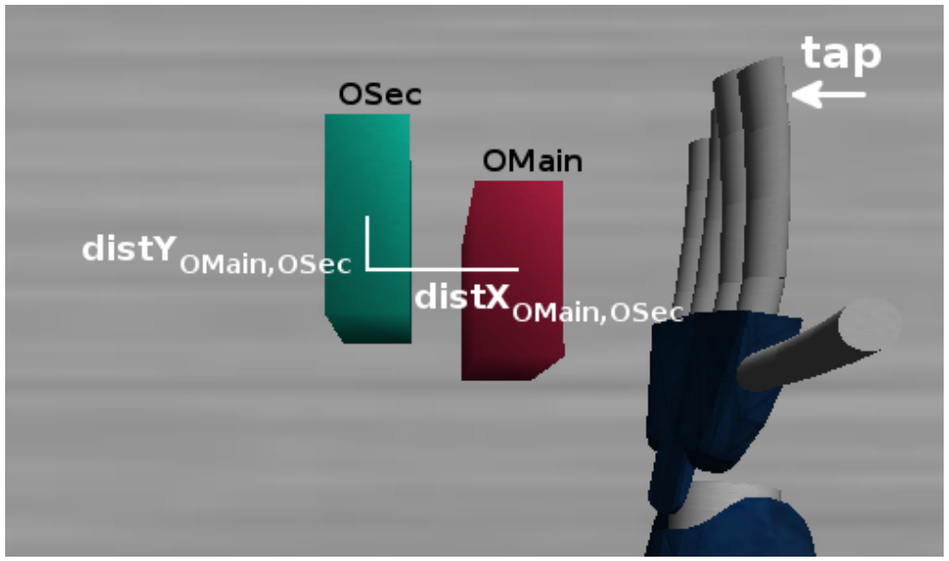
Learning relational affordances between two objects (learnt by experience)

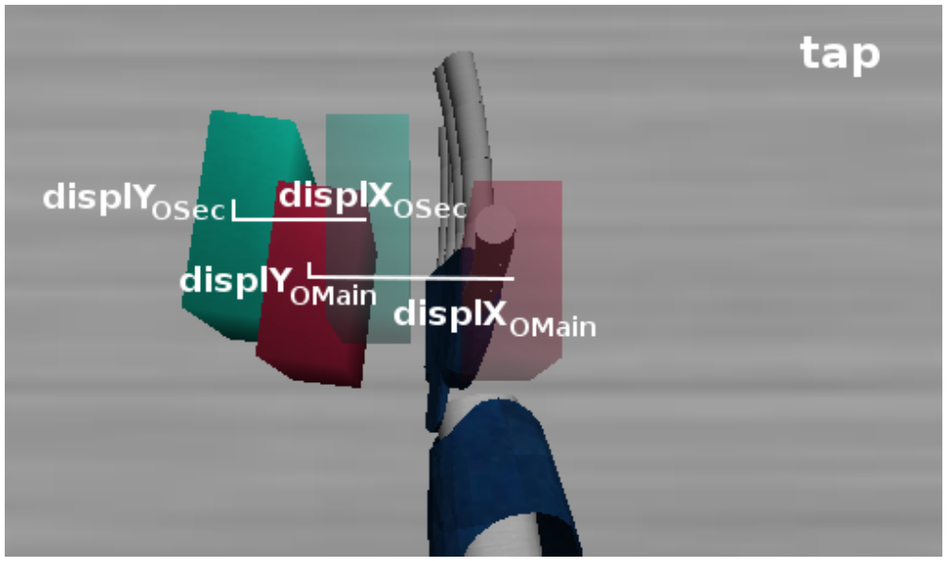
From two object interactions

Generalize to N

***Moldovan** et al. ICRA 12, 13, 14, PhD 15*

What is an affordance ?





Object Properties	Action	Effects
$shape_{O_{Main}} : sprism$ $shape_{O_{Sec}} : sprism$ $distX_{O_{Main},O_{Sec}} : 6.94cm$ $distY_{O_{Main},O_{Sec}} : 1.90cm$	$tap(10)$	$displX_{O_{Main}} : 10.33cm$ $displY_{O_{Main}} : -0.68cm$ $displX_{O_{Sec}} : 7.43cm$ $displY_{O_{Sec}} : -1.31cm$

- Formalism — related to STRIPS but models delta
- but also joint probability model over A, E, O

Learning relational affordances between two objects (learnt by experience)

Right Arm

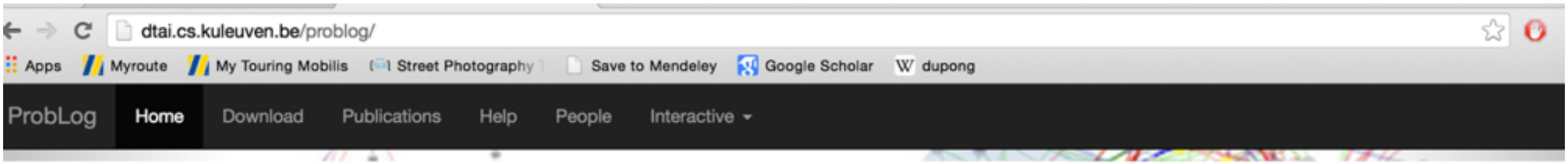
Examples

Learning relational affordances between two objects (learnt by experience)

Right Arm

Examples

ProbLog



Introduction.

Probabilistic logic programs are logic programs in which some of the facts are annotated with probabilities.

ProbLog is a tool that allows you to intuitively build programs that do not only encode **complex interactions** between a large sets of **heterogenous components** but also the inherent **uncertainties** that are present in real-life situations.

The engine tackles several tasks such as computing the marginals given evidence and learning from (partial) interpretations. ProbLog is a suite of efficient algorithms for various inference tasks. It is based on a conversion of the program and the queries and evidence to a weighted Boolean formula. This allows us to reduce the inference tasks to well-studied tasks such as weighted model counting, which can be solved using state-of-the-art methods known from the graphical model and knowledge compilation literature.

The Language. Probabilistic Logic Programming.

ProbLog makes it easy to express complex, probabilistic models.

```
0.3::stress(X) :- person(X).
0.2::influences(X,Y) :- person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :- friend(X,Y), influences(Y,X), smokes(Y).

0.4::asthma(X) <- smokes(X).

person(angelika).
person(joris).
person(jonas).
person(dimitar).

friend(joris,jonas).
```

Probabilistic Programs

- Distributional clauses similar in spirit to probabilistic functional languages such as
  - BLOG [Russell et al.], ... but embedded in existing logic and programming language
  - Church [Goodmann et al.] but use of logic instead of functional programming ...
- Probabilistic Logic Programming (survey: De Raedt & Kimmig, MLJ 15):
  - natural possible world semantics and link with prob. databases.
  - somewhat harder to do meta-programming

Probabilistic Programming

Key idea :

- modeling / programming language
- extend with probabilistic primitives to define probability distribution over possible worlds or execution traces
- extend with solvers / execution strategies to answer probabilistic queries (marginal, conditional probabilities, MAP and MPE)
- extend with learning strategies (EM or Bayesian inference) to estimate parameters and/or learn structure

Many other probabilistic programming languages including

- PRISM (Sato), Pita (Riguzzi), Blog (Russell), Figaro (Pfeffer), Church (Goodmann), Factorie (McCallumn), Anglican (Wood), Venture (Mansinghka), …

Programming Languages for Machine Learning

Kernel Programming

with Paolo Frasconi, Francesco Orsini et al.

Machine learning

Given

- an unknown target function f: X → Y
- a hypothesis space L containing functions X → Y
- a dataset of examples E = { (x, f(x)) | x ∈ X }
- a loss function loss(h,E) → $\mathbb{R}$

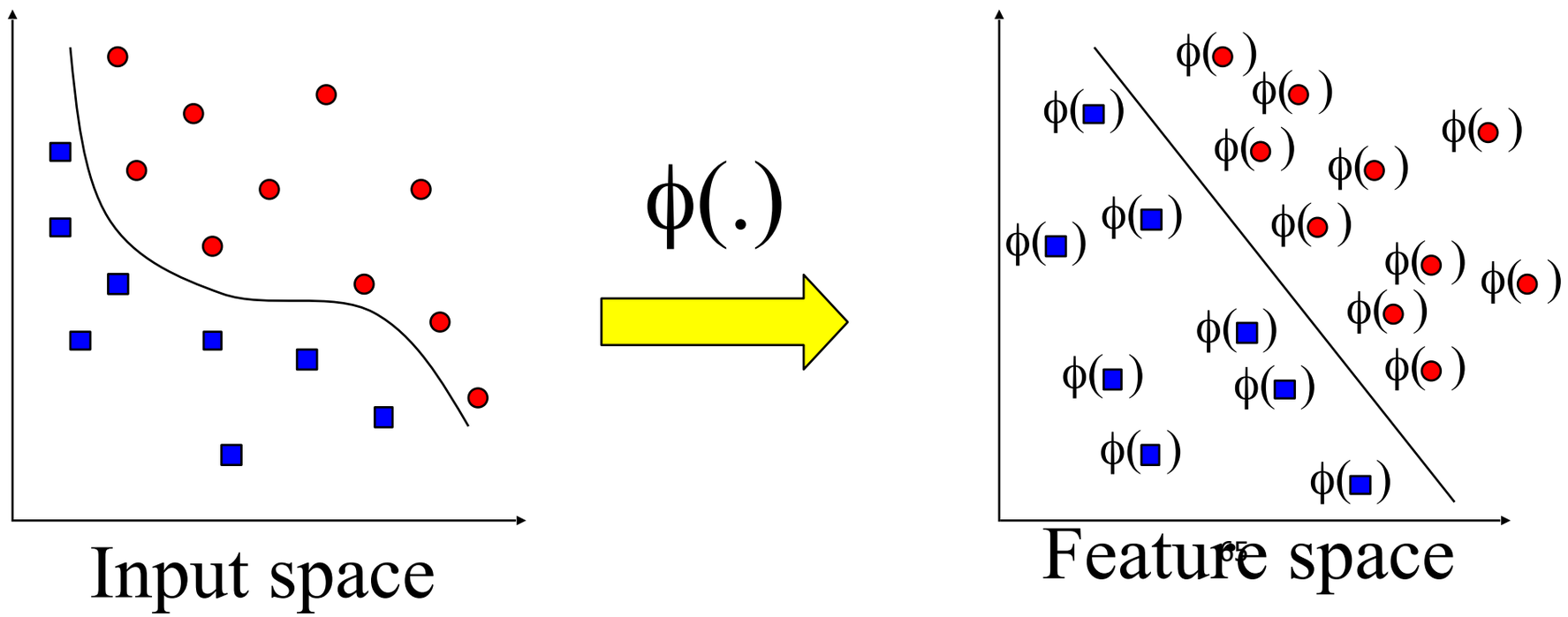
supervised

Find h ∈ L that minimizes loss(h,E)

Kernels and SVMs

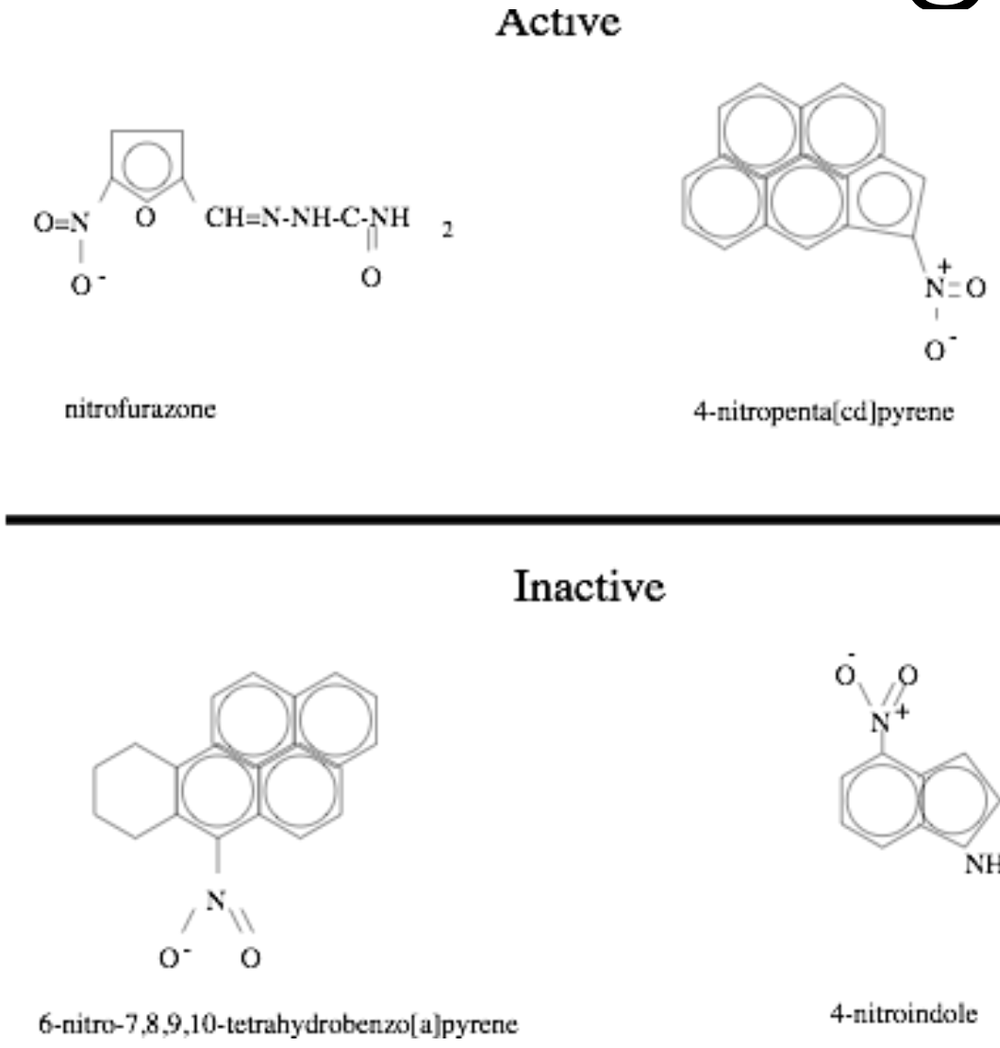
- A kernel is a similarity measure between two "observations", a dot product in a feature space

- Kernels are used by SVMs to efficiently compute a linear separator $k(x, x') =< \phi(x), \phi(x') >$



Inductive Logic Programming

[Srinivasan et al. AIJ 96]



Data = Set of Small Graphs

General Purpose
Logic Learning System

Uses and Produces
Knowledge

Learning from entailment

```
molecule(225).
logmutag(225,0.64).
lumo(225,-1.785).
logp(225,1.01).
nitro(225,[f1_4,f1_8,f1_10,f1_9]).
atom(225,f1_1,c,21,0.187).
atom(225,f1_2,c,21,-0.143).
atom(225,f1_3,c,21,-0.143).
atom(225,f1_4,c,21,-0.013).
atom(225,f1_5,o,52,-0.043).
...
ring_size_5(225,[f1_5,f1_1,f1_2,f1_3,f1_4]).
hetero_aromatic_5_ring(225,[f1_5,f1_1,f1_2,f1_3,f1_4]).
```

background

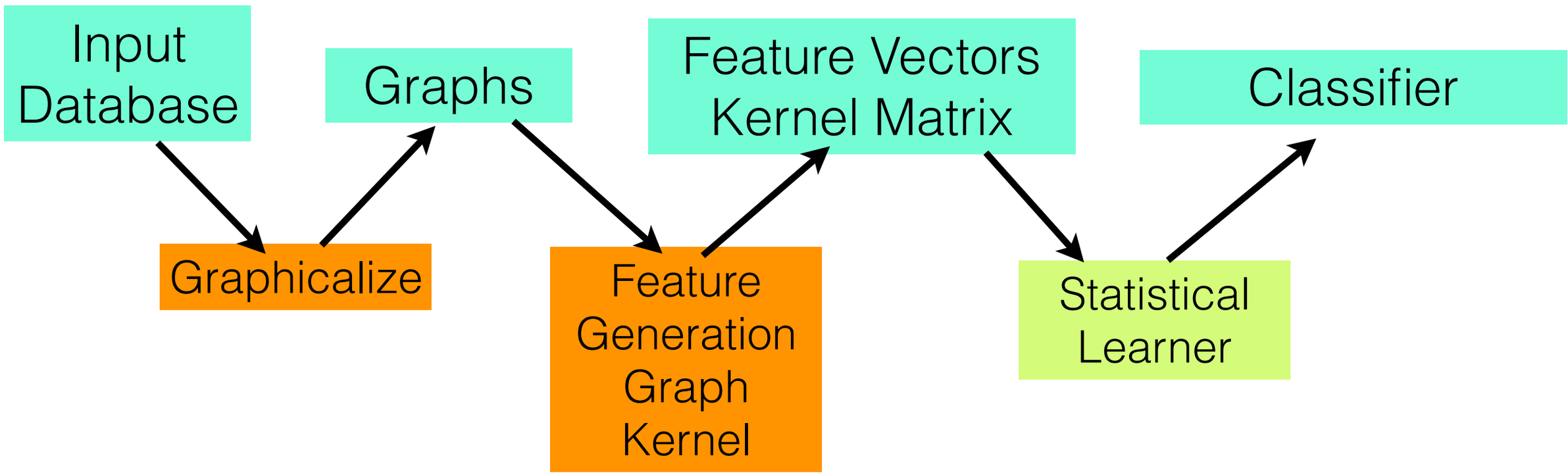
```
bond(225,f1_1,f1_2,7).
bond(225,f1_2,f1_3,7).
bond(225,f1_3,f1_4,7).
bond(225,f1_4,f1_5,7).
bond(225,f1_5,f1_1,7).
bond(225,f1_8,f1_9,2).
bond(225,f1_8,f1_10,2).
bond(225,f1_1,f1_11,1).
bond(225,f1_11,f1_12,2).
bond(225,f1_11,f1_13,1).
```

`mutagenic(225), ...` **examples**

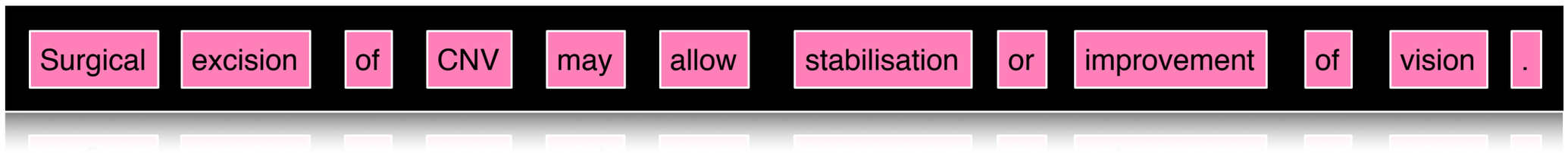
how to define a kernel between relational examples ?

how to do that declaratively ?

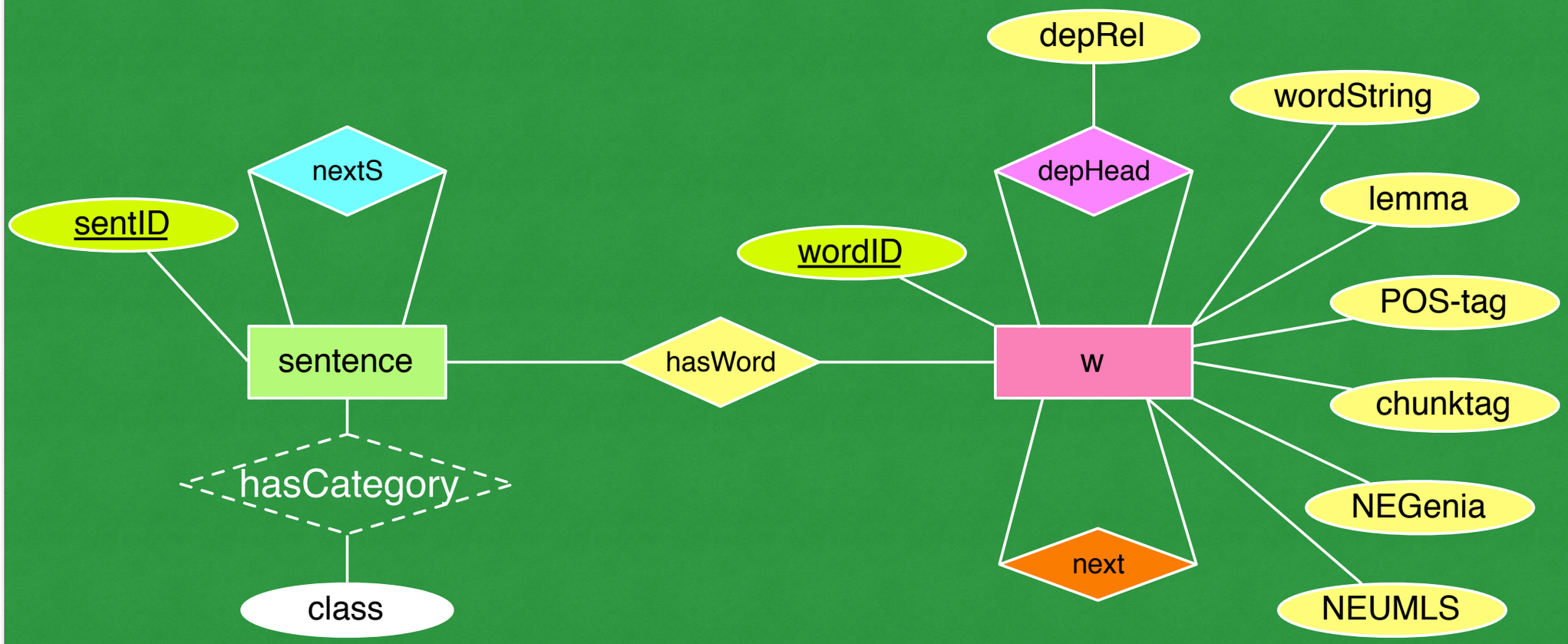
kLOG [Frasconi et al AIJ 14]



A biomedical NLP task [Verbeke et al. EMLNP 12]



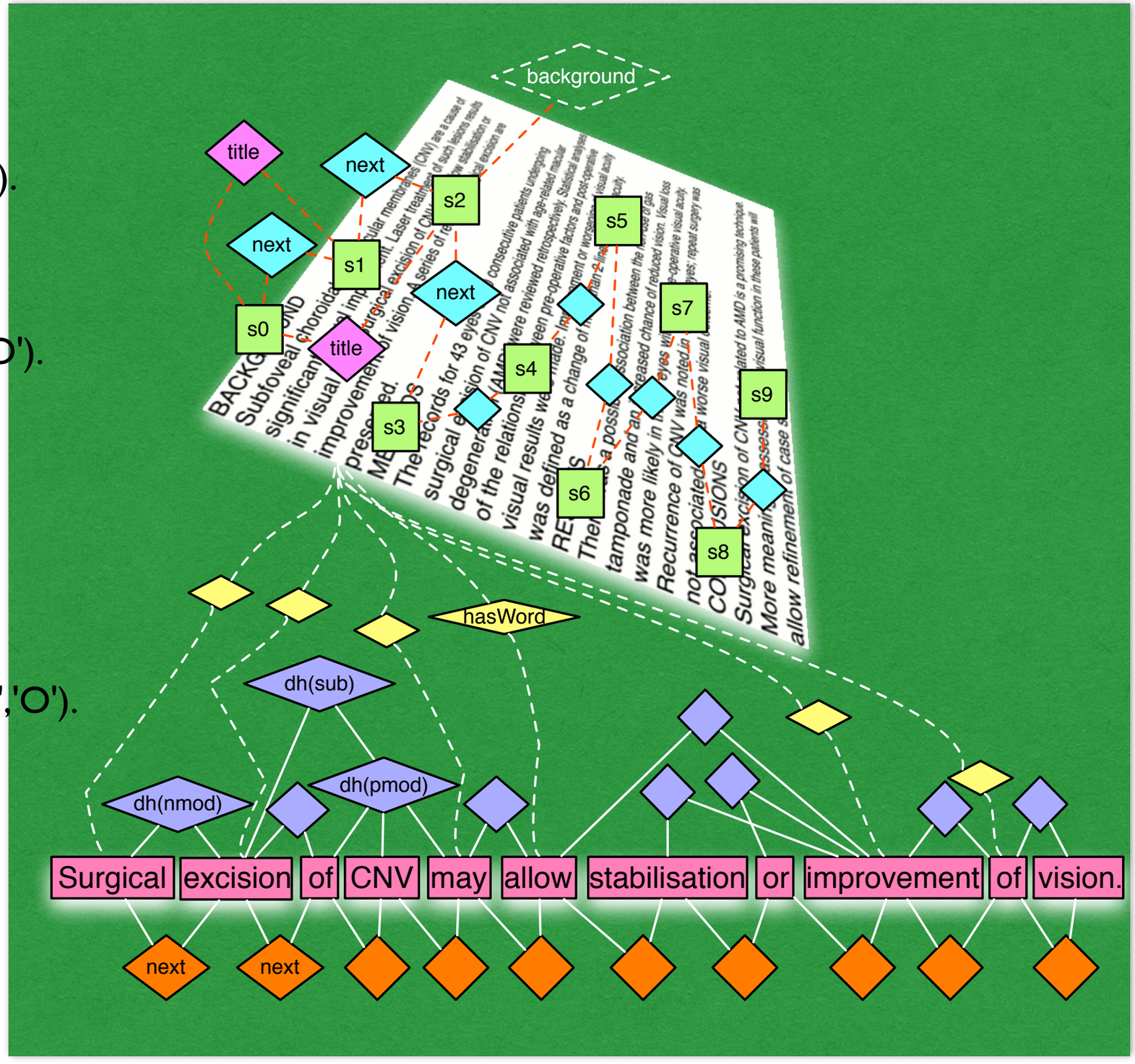
E/R-MODEL



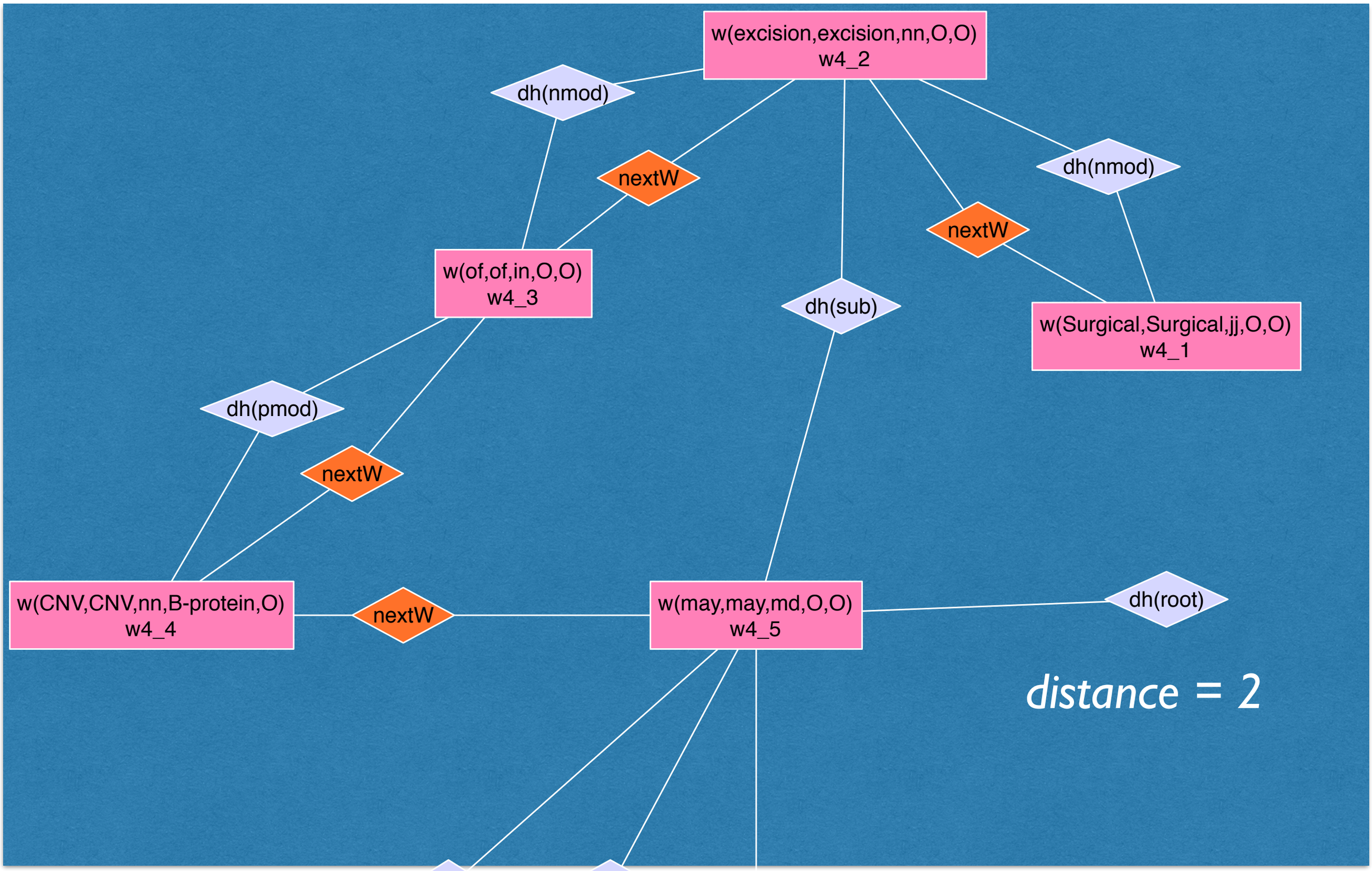
[Verbeke et al. EMNLP 12]

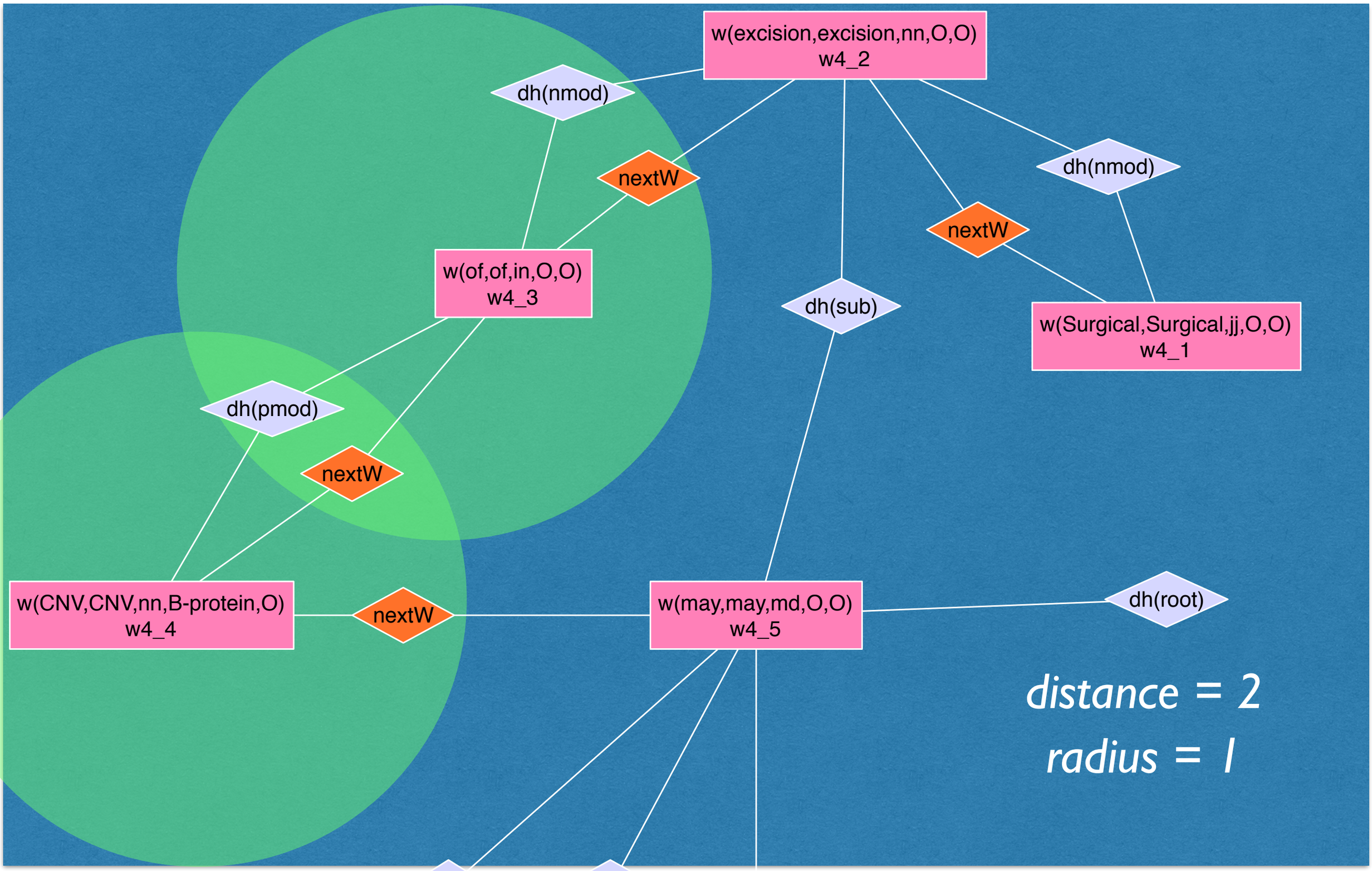
```
sentence(s4,4).
hasCategory(s4,'background').
w(w4_1,'Surgical','Surgical',b-np,jj,'O','O').
hasWord(s4,w4_1).
dh(w4_1,w4_2,nmod).
nextW(w4_2,w4_1).
w(w4_2,'excision','excision',i-np,nn,'O','O').
hasWord(s4,w4_2).
dh(w4_2,w4_5,sub).
nextW(w4_3,w4_2).
w(w4_3,'of','of',b-pp,in,'O','O').
hasWord(s4,w4_3).
dh(w4_3,w4_2,nmod).
nextW(w4_4,w4_3).
w(w4_4,'CNV','CNV',b-np,nn,'B-protein','O').
hasWord(s4,w4_4).
dh(w4_4,w4_3,pmod).
nextW(w4_5,w4_4).
w(w4_5,'may','may',b-vp,md,'O','O').
hasWord(s4,w4_5).
dh(w4_5,w4_0,root).
nextW(w4_6,w4_5).
```

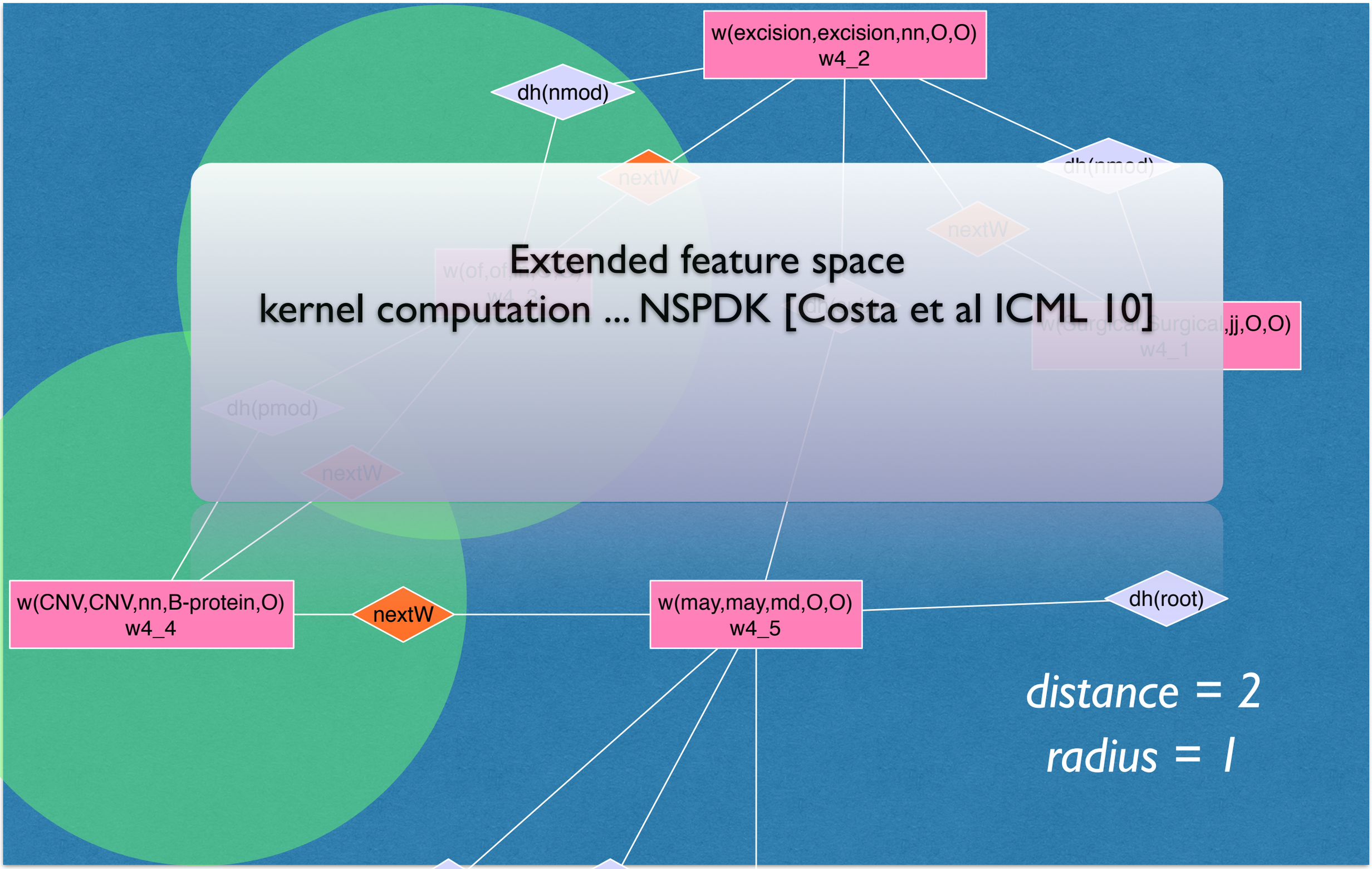
...

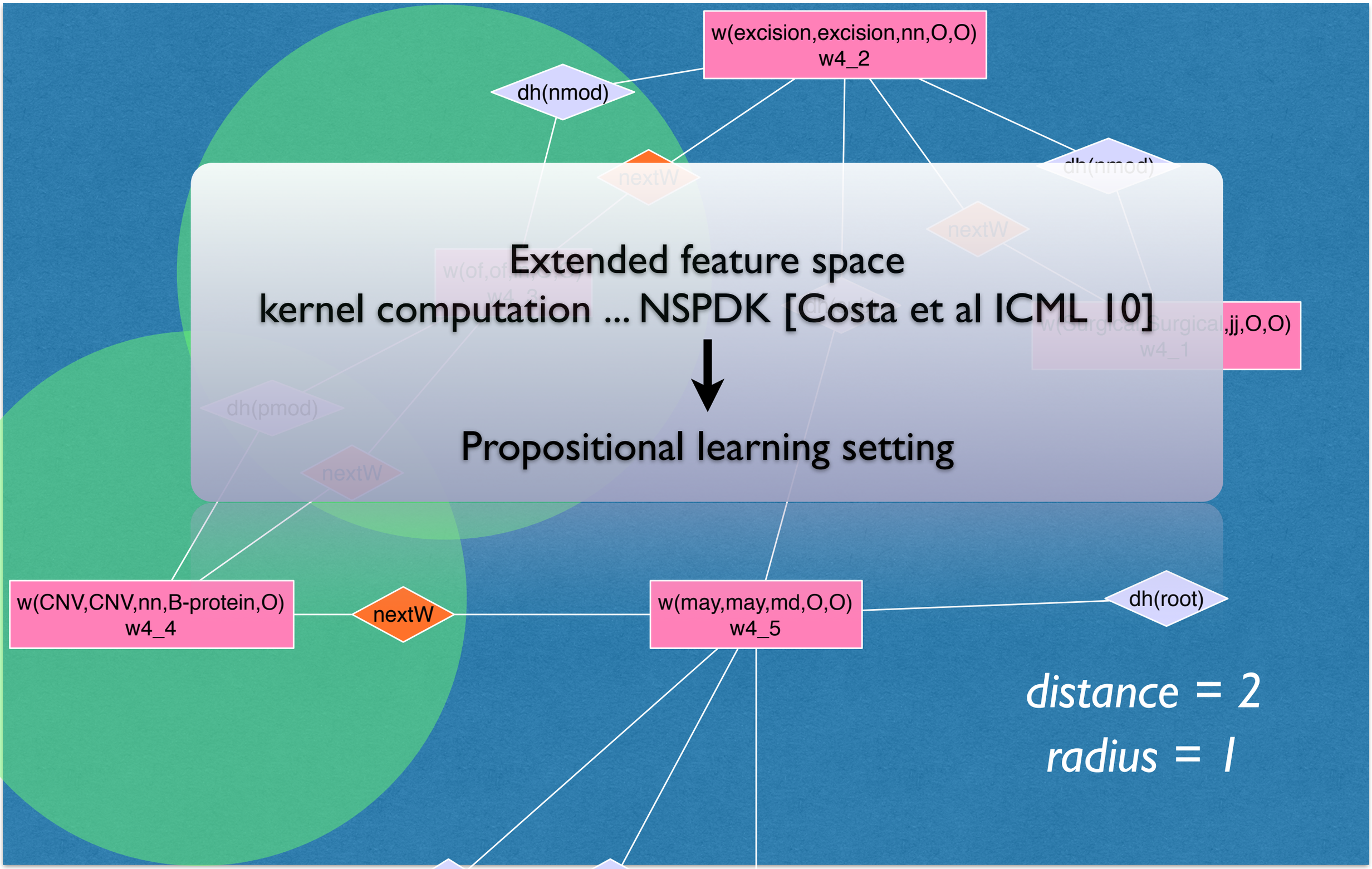


Graphicalization

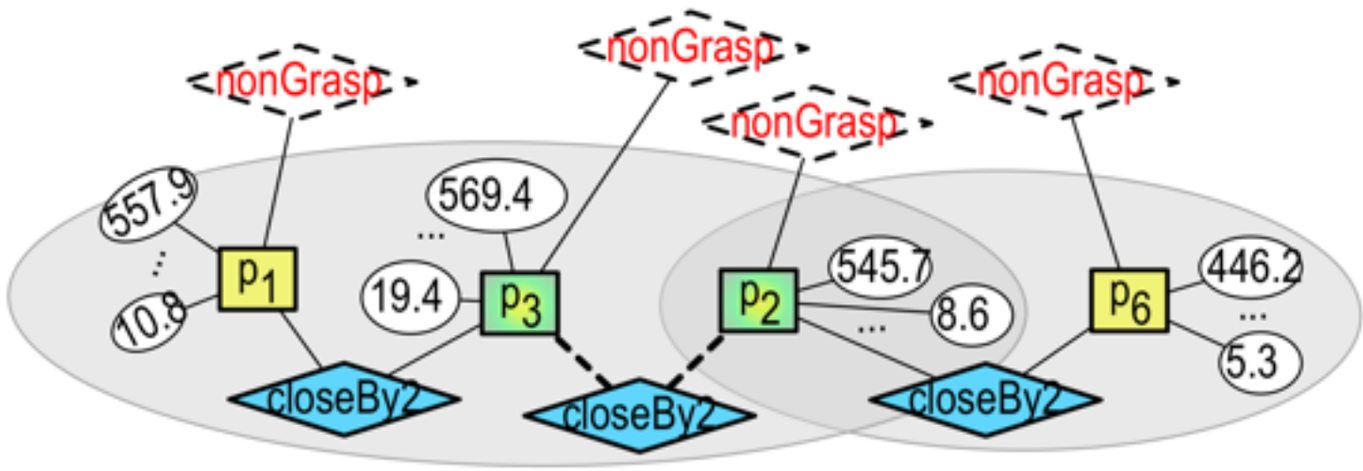

w(excision,excision,nn,O,O)
w4_2
dh(nmod)
nextW
dh(nmod)
nextW
w(of,of,in,O,O)
w4_3
dh(sub)
w(Surgical,Surgical,jj,O,O)
w4_1
dh(pmod)
nextW
w(CNV,CNV,nn,B-protein,O)
w4_4
nextW
w(may,may,md,O,O)
w4_5
dh(root)
distance = 2


w(excision,excision,nn,O,O)
w4_2
dh(nmod)
nextW
dh(nmod)
nextW
w(of,of,in,O,O)
w4_3
dh(sub)
w(Surgical,Surgical,jj,O,O)
w4_1
dh(pmod)
nextW
w(CNV,CNV,nn,B-protein,O)
w4_4
nextW
w(may,may,md,O,O)
w4_5
dh(root)
distance = 2
radius = 1


w(excision,excision,nn,O,O)
w4_2
dh(nmod)
nextW
dh(nmod)
nextW
dh(pmod)
nextW
w(CNV,CNV,nn,B-protein,O)
w4_4
nextW
w(may,may,md,O,O)
w4_5
dh(root)
Extended feature space
kernel computation ... NSPDK [Costa et al ICML 10]
distance = 2
radius = 1


w(excision,excision,nn,O,O)
w4_2
dh(nmod)
nextW
dh(nmod)
nextW
w(Surgical,Surgical,jj,O,O)
w4_1
dh(pmod)
nextW
Extended feature space
kernel computation ... NSPDK [Costa et al ICML 10]
Propositional learning setting
w(CNV,CNV,nn,B-protein,O)
w4_4
nextW
w(may,may,md,O,O)
w4_5
dh(root)
distance = 2
radius = 1

Graph Kernels



The decomposition kernel is defined by relations $R_{r,d}$:

$$K(G,G') = \sum_{r=0}^{R} \sum_{d=0}^{D} \sum k((A,B),(A',B')).$$

A,B: $R_{r,d}^{-1}(A,B,G)$
A',B': $R_{r,d}^{-1}(A',B',G')$

- k((A,B),(A′,B′)) = 1 iff (A,B) and (A′,B′) are pairs of isomorphic subgraphs — hard match kernel
- k((A,B),(A′,B′)): multinomial distribution of labels in (A,B) or (A’,B’) — soft match kernel

klog.dinfo.unifi.it

kLog | Home | Download | Data sets | Publications | Tutorials | Documentation | Contacts

kLog

A language for relational learning with kernels

kLog is a logical and relational language for kernel-based learning. Logical and relational learning problems may be specified at a high level in a declarative way. It builds on simple but powerful concepts: learning from interpretations, entity/relationship data modeling, logic programming and deductive databases (Prolog and Datalog), and graph kernels.

Unlike other statistical relational learning models, kLog does not represent a probability distribution directly. It is rather a kernel-based approach to learning that employs features derived from a grounded entity/relationship diagram. These features are derived using a novel technique called graphicalization: first, relational representations are transformed into graph based representations; subsequently, graph kernels are employed for defining feature spaces. kLog can use numerical and symbolic data, background knowledge in the form of Prolog or Datalog programs (as in inductive logic programming systems) and several statistical procedures can be used to fit the model parameters. The kLog framework can --- in principle --- be applied to tackle the same range of tasks that has made statistical relational learning so popular, including classification, regression, multitask learning, and collective classification.

Checkout kLogNLP, a specialized version of kLog for natural language processing

Graph kernels vs kProlog

learning with linear separators

feature vectors

machine learning

graph kernel

Φ

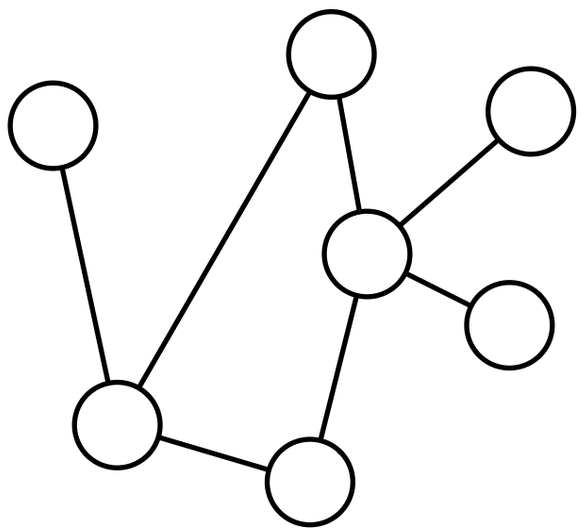
algebraic labels meta-functions

Φ

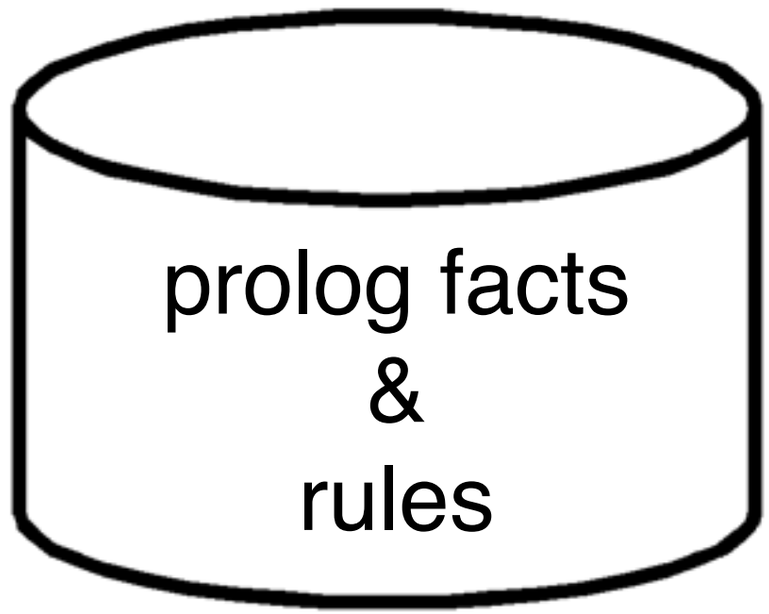
polynomials for feature extraction

domain representation

graph



knowledge base



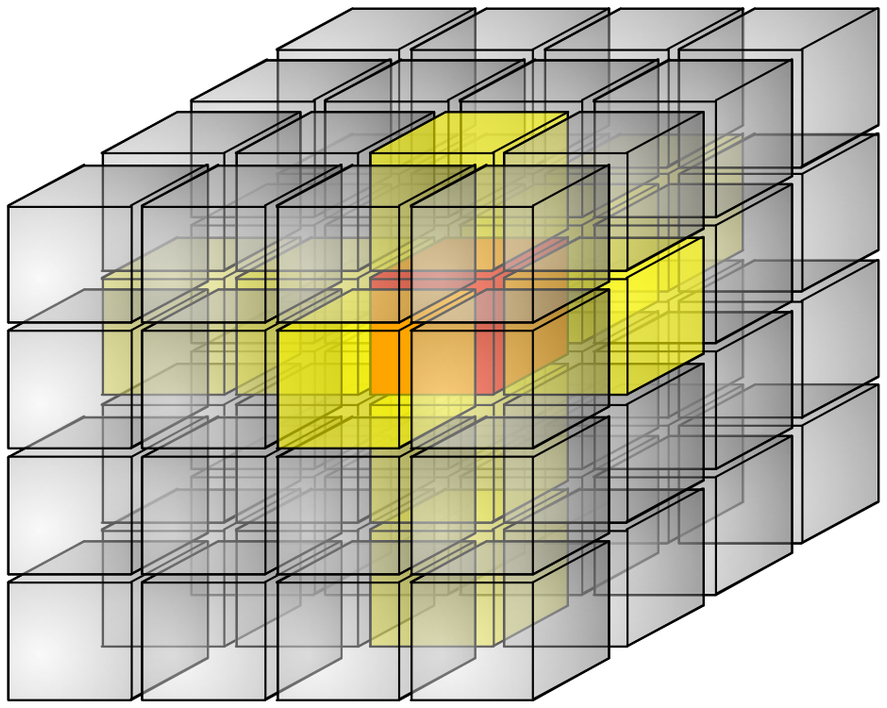
Tensor operations

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$$

```
:- declare(a/2, int).
1::a(0,0).
2::a(0,1).
3::a(1,1).
```

$$B = \begin{bmatrix} 2 & 1 \\ 5 & 1 \end{bmatrix}$$

```
:- declare(b/2, int).
2::b(0,0).
1::b(0,1).
5::b(1,0).
1::b(1,1).
```



semi-rings ! cf. Dyna [Eisner]
aProbLog [Kimmig]

transpose A^t

```
c(I, J):-
    a(J, I).
```

addition $A + B$

```
c(I, J):- a(I, J).
c(I, J):- b(I, J).
```

matrix product AB

```
c(I, J):-
    a(I, K), b(K, J).
```

and more…

kProlog$^{S}[\mathbf{x}]$

some relevant operations

sum $\oplus$

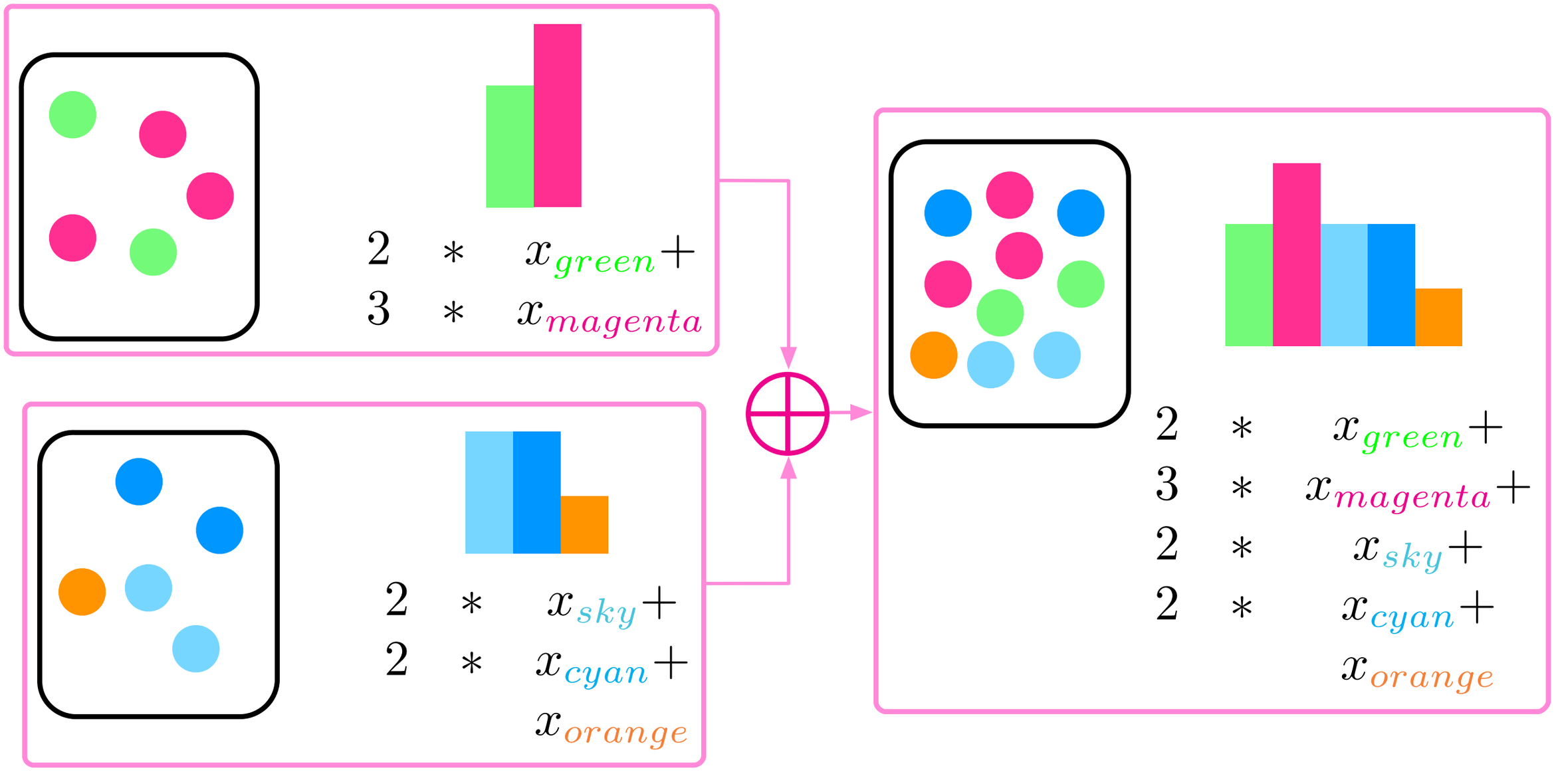
compress $@id$

dot product $@dot$

[Orsini, ILP 15]

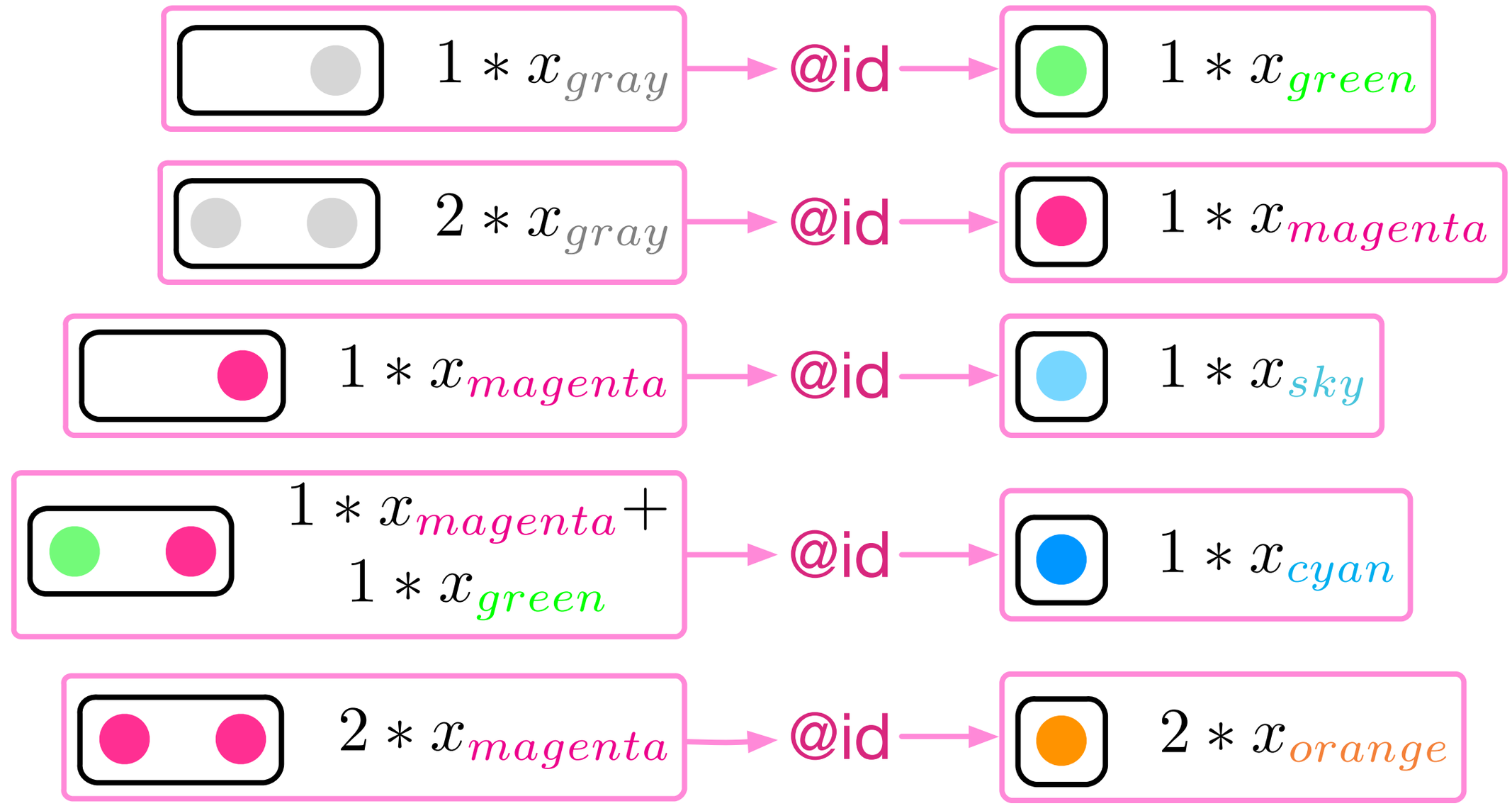
kProlog$^{S[\mathbf{x}]}$

semiring sum = feature addition



kProlog$^{S}[\mathbf{x}]$

@id function = feature compression



analogous of the f function in [Shervashidze et al. (2011)]

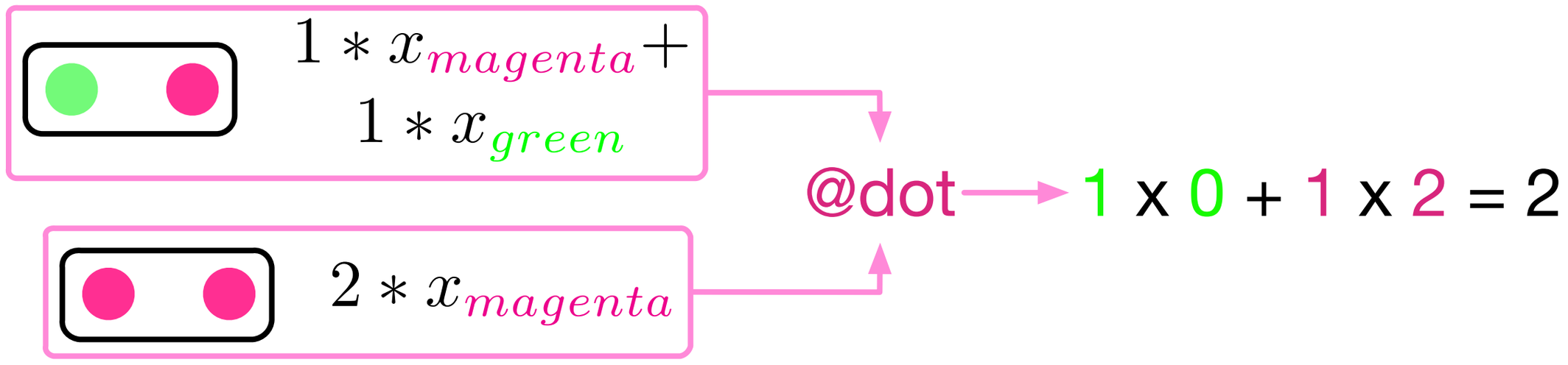
kProlog$^{S[\mathbf{x}]}$

@dot product

$$\langle \mathcal{P}(\mathbf{x}), \mathcal{Q}(\mathbf{x}) \rangle = \sum_{(p,\mathbf{e})\in\mathcal{P}} \sum_{(q,\mathbf{e})\in\mathcal{Q}} pq$$

---

example

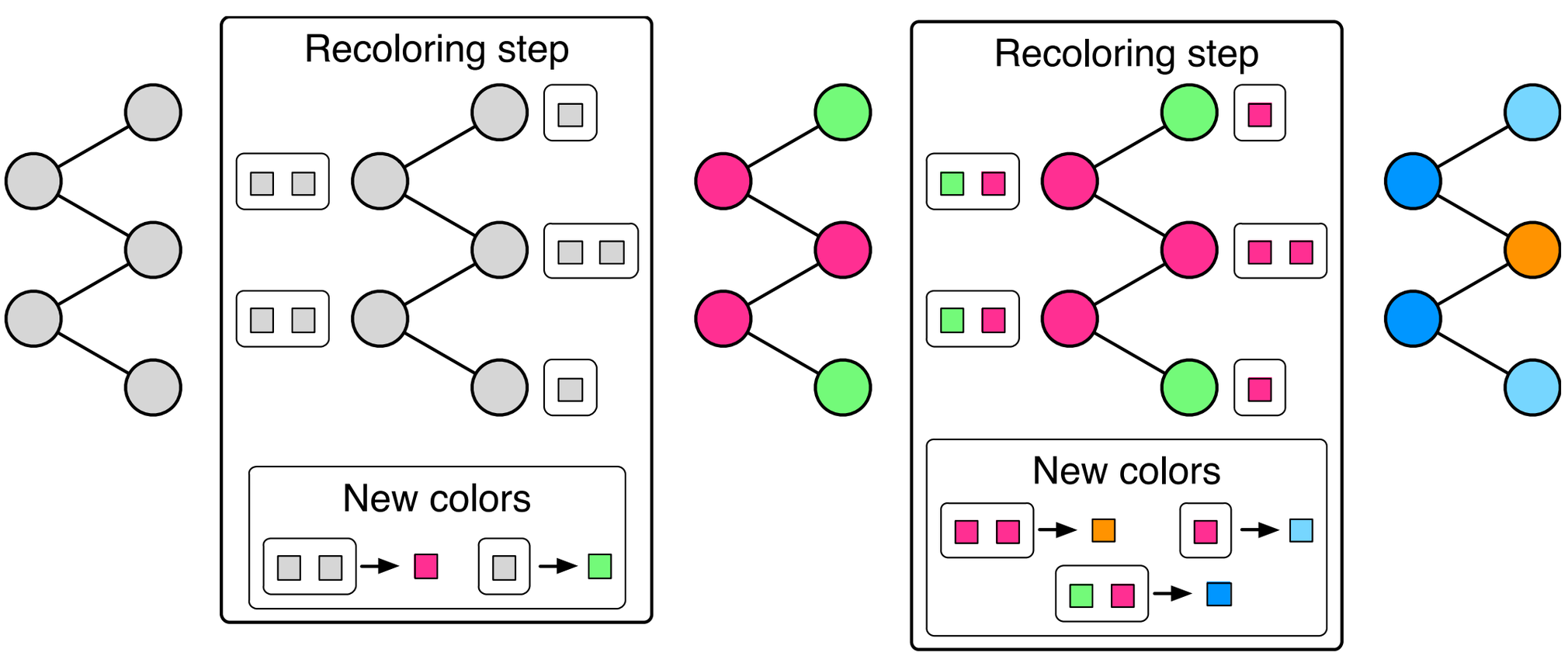


Weisfeiler-Lehman algorithm

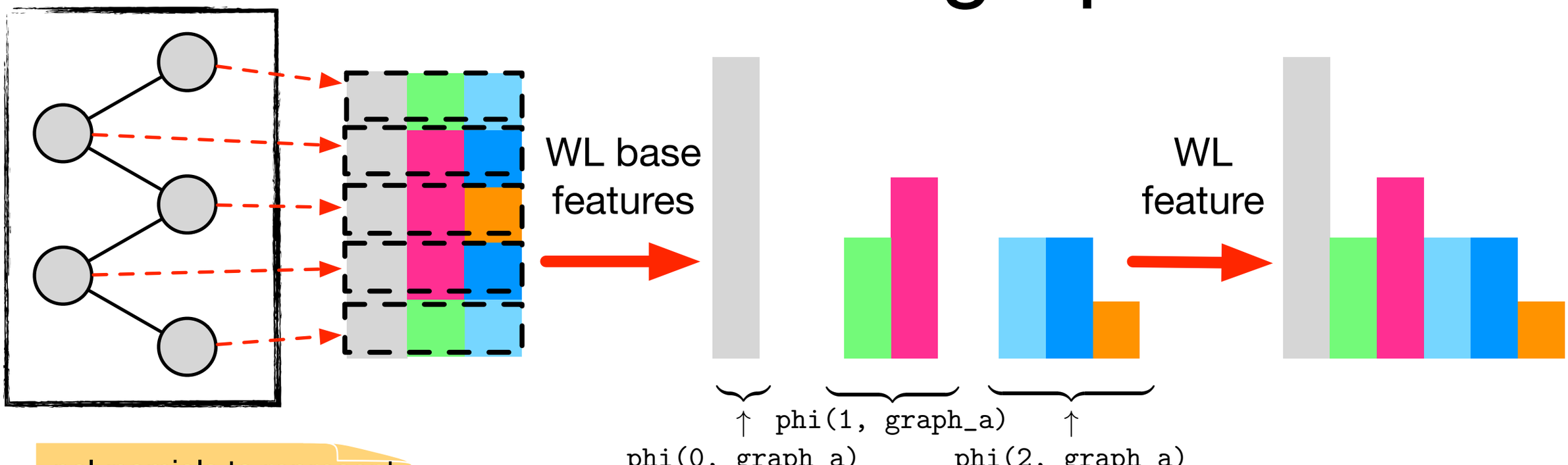
(a.k.a. color refinement)

Can also be used to initialise GI-testing algorithms.

$$\mathcal{L}^h(v) = \begin{cases} \ell(v) & \text{if } h = 0 \\ f(\{\mathcal{L}^{h-1}(w) | w \in \mathcal{N}(v)\}) & \text{if } h > 0 \end{cases}$$



Weisfeiler-Lehman graph kernel

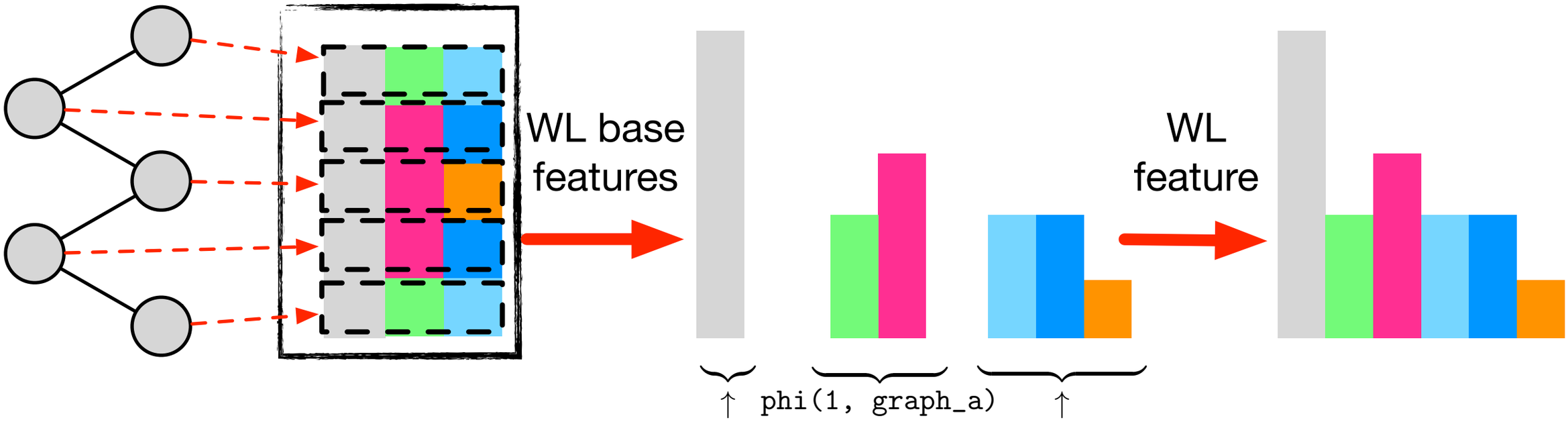


polynomials to represent graph labels

```
:- declare(vertex/2, polynomial(int)).
:- declare(edge_asymm/3, boolean).
:- declare(edge/3, polynomial(int)).

1 * x(gray)::vertex(graph_a, 1).
1 * x(gray)::vertex(graph_a, 2).
1 * x(gray)::vertex(graph_a, 3).
1 * x(gray)::vertex(graph_a, 4).
1 * x(gray)::vertex(graph_a, 5).
```

```
edge_asymm(graph_a, 1, 2).
edge_asymm(graph_a, 2, 3).
edge_asymm(graph_a, 3, 4).
edge_asymm(graph_a, 4, 5).

1.0::edge(Graph, A, B):-
    edge_asymm(Graph, A, B).

1.0::edge(Graph, A, B):-
    edge_asymm(Graph, B, A).
```

Weisfeiler-Lehman graph kernel



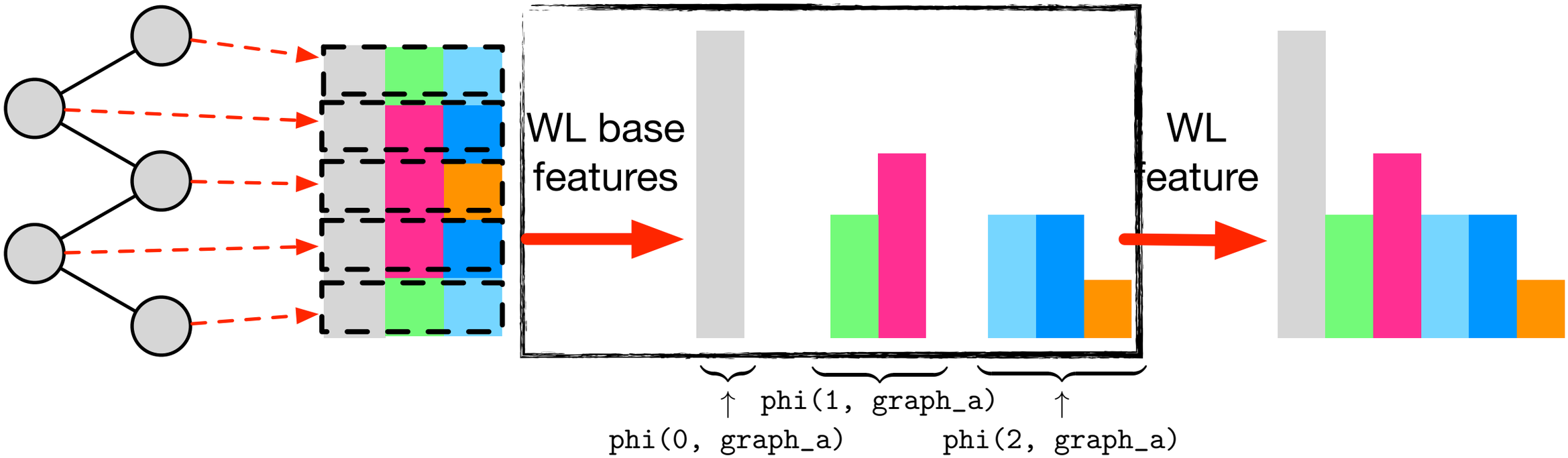

```
:- declare(wl_color/3,
     polynomial(int)).
:- declare(wl_color_multiset/3,
     polynomial(int)).

wl_color_multiset(H, Graph, V):-
  edge(Graph, V, W),
  wl_color(H, Graph, W).
```

polynomials represent multisets of labels

```
wl_color(0, Graph, V):-
  vertex(Graph, V).

wl_color(H, Graph, V):-
  H > 0,
  H1 is H - 1,
  @id[wl_color_multiset(H1, Graph, V)].
```

@id meta-function for recoloring

Weisfeiler-Lehman graph kernel



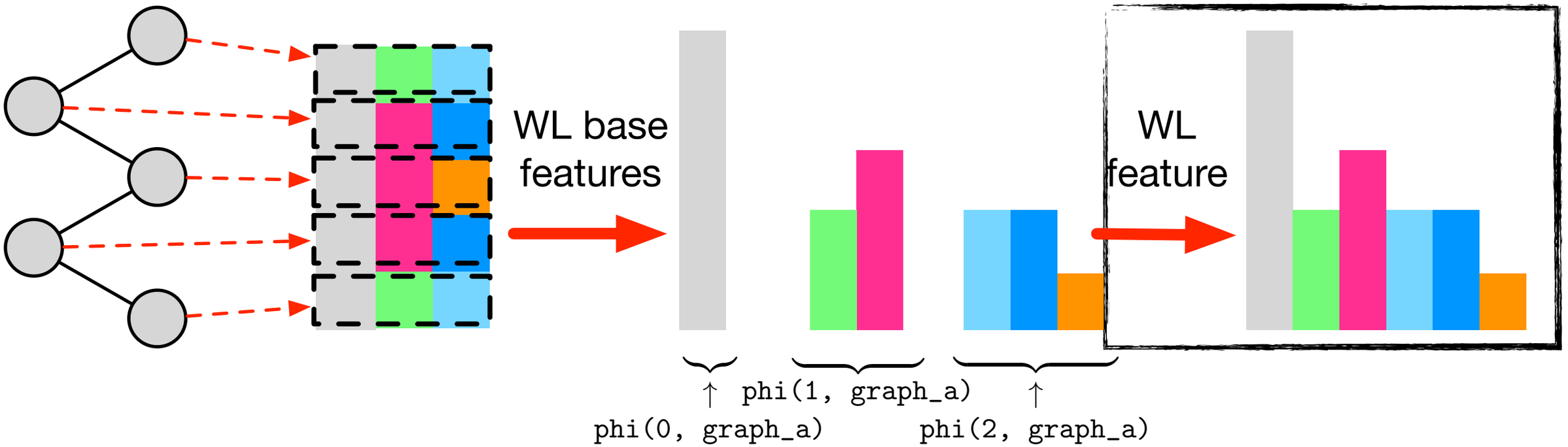

```
:- declare(phi/2, real).
phi(H, Graph):-
  wl_color(H, Graph, V).
```

explicit feature vector at iteration H

```
:- declare(base_kernel/3, real).
base_kernel(H, Graph, GraphPrime):-
  @dot[phi(H, Graph),
       phi(H, GraphPrime)].
```

the base kernel H is the dot product between explicit feature vector at iteration H

Weisfeiler-Lehman graph kernel




```
:- declare(kernel_wl/3, real).


kernel_wl(0, Graph, GraphPrime):-
  base_kernel(0, Graph, GraphPrime).
```

```
kernel_wl(H, Graph, GraphPrime):-
  H > 0, H1 is H - 1,
  kernel_wl(H1, Graph, GraphPrime).

kernel_wl(H, Graph, GraphPrime):-
  H > 0,
  base_kernel(H, Graph, GraphPrime).
```

accumulate base-kernels of successive iterations

kProlog

- Based on the idea of semi-rings (akin to Dyna [Eisner] and aProbLog [Kimmig])
- Combines several semi-rings, employs meta-functions; semantics based on Tp-operator.
- Can be used for declarative “kernel programming”
- [Orsini ILP 15]

Role of Declarative Methods in ML and DM ?

- Two way interaction
  - Many opportunities for ML/DM
    - expressive power, ease of modeling, solver independent
  - New challenges for Declarative Methods
- Many open issues and opportunities for research

…

Thanks !